LEARNING CONDITIONAL MODELS FOR VISUAL PERCEPTION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Andreas Veit

May 2018

LEARNING CONDITIONAL MODELS

FOR VISUAL PERCEPTION

Andreas Veit, Ph.D.

Cornell University 2018

In recent years, the field of computer vision has seen a series of major advances, made possible by rapid development in algorithms, data collection and computing infrastructure. As a result, vision systems have started to be broadly adopted in everyday applications. Progress has been particularly promising in image recognition, where algorithms now often match human performance. Nevertheless, vision systems still largely fall behind humans in their ability to understand the complexities of the visual world and its apparent contradictions. For example, an image can carry different meanings to different people in different contexts. However, being often limited to a single point of view, vision systems tend to focus on the meaning that dominates in the training data.

In this dissertation, we address this limitation by building conditional vision models that can learn from multiple points of view and adapt their results to account for different conditions. First, we address the related tasks of image tagging and tag based image retrieval. In particular, we build a system that can take into account the fact that people may associate different meaning with certain images and tags. Thus, the system can personalize outputs for ambiguous tags such as #rock, which could refer either to a music genre, a geological object or even outdoor climbing. Further, we focus on the task of image based similarity search. Specifically, we design a system that can understand multiple notions of similarity. For example, when searching for related items to an input

images of a shoe, users might be interested in shoes of similar color, style, or for the same kind of activity. By capturing the multitude of aspects in terms of which objects can be compared, our system can find the right set of related items. Lastly, we explore how the underlying convolutional networks themselves can be made aware of the context in which they are used. In a study, we first discover a new understanding of the roles that individual layers take on in modern convolutional networks. Then, we leverage our insights and design a network that can adaptively define its own topology conditioned on the input image to increase both accuracy and efficiency.

BIOGRAPHICAL SKETCH

Andreas Veit is a PhD Candidate in Computer Science at Cornell University working with Prof. Serge Belongie. His research focuses on Computer Vision and Machine Learning. Before joining Cornell, he obtained his B.Sc and M.Sc in Information Engineering and Management from Karlsruhe Institute of Technology (KIT), Germany, in 2010 and 2013 respectively. During his studies, Andreas spent visiting semesters at KTH in Stockholm, University of Connecticut as well as Carnegie Mellon University. While at Cornell, he was supported by an Oath PhD fellowship and his studies at KIT were supported by scholarships from Cusanuswerk, InterAct and the Baden-Württemberg Foundation. Andreas is recipient of the award for an outstanding bachelor degree and fellow of the college for gifted students both of the Department of Informatics at KIT.

To my parents Rita and Jürgen.

ACKNOWLEDGEMENTS

There are many people I would like to thank for contributing to the great experience of my PhD journey. Foremost, I want to express my gratitude to my advisor Serge Belongie, for his mentorship, support and confidence in me throughout my years at Cornell. Serge helped me to become an independent researcher, by always giving me the space, freedom and encouragement to develop and pursue my own ideas. Furthermore, he was always present when I needed guidance both on academic as well as non-academic matters.

I would also like to thank my thesis committee members Mor Naaman and Jon Kleinberg for their invaluable suggestions. Their questions always provided different new perspectives and encouragement.

Further, I want to thank Hartmut Schmeck and Katia Sycara for advising me during my master studies and for introducing me to academic research. It was due to my stay in Katia's lab at CMU that I decided to pursue a PhD.

I've been very fortunate to learn from and work with many remarkably inspiring colleagues. Without my lab mates Michael Wilber, Hani Altwaijry and Yin Cui and my close collaborators Theofanis Karaletsos from MSKCC, Balazs Kovacs from Cornell and David Rolnick from MIT, the past few years would not have been such a great experience and much of this work would not have been possible. I also want to extend gratitude to my faculty collaborators Kavita Bala, Julian McAuley and Abhinav Gupta for their advice and support on various projects.

Besides my close collaborators, a number of other friends and colleagues have been a source of many interesting discussions and research ideas. In particular, Tsung-Yi Lin, Baoguang Shi, Vlad Niculae, Xiying Wang, Xun Huang, Sean Bell, Omid Poursaeed, Eugene Bagdasaryan from Cornell, Ishan Misra

from CMU, Ramakrishna Vedantam from Georgia Tech, Sam Kwak from UCSD, Lucas Czech from HITS and Rajan Vaish from UCSC.

During the summers, I had the opportunity to intern with great researchers including Neil Alldrin, Gal Chechik, Laurens van der Maaten and Maximilian Nickel, whose advice helped me well beyond the summers.

I am thankful to several funding sources that supported my research, including a Facebook Research Award, a Google Focused Research Award and the Oath Program for Connected Experiences at Cornell Tech.

Thanks also to my dear friends Georg Arndt and Gargi Wable for their moral support during these years. A special thanks goes to Antonio Marcedone for being the best roommate and for always being there for me.

Finally, my deepest gratitude goes to my parents Rita and Jürgen, my sister Alexandra and my partner Yiqing Hua. Having them at my side has been a tremendous source of strength. Thank you for all the love, enthusiasm and support during this wonderful journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A longtime goal in computer vision is the development of methods that can achieve human-level understanding of the visual world. While there is still a long way to go, great progress has been made in the last years thanks to advances in algorithms, data collection and computing infrastructure. Particularly impactful have been advances in image classification. Computers can now classify images into thousands of categories with accuracy similar to humans. Sometimes they even surpass human performance for tasks such as identifying different bird species. However, in spite of encouraging advances, modern computer vision systems largely lack the ability to understand the visual world from multiple different perspectives. While for humans contradicting answers can co-exist depending on the context, computer vision systems regularly lack the ability to capture contradictions. As a consequence, they often need to choose a single point of view and in practice tend to provide the answer that dominates in the training set. This becomes a challenge for many tasks such as, for instance, hashtag-based image retrieval, where different hashtags might carry different meaning for different people. Consider the example of searching for the hashtag #rock; some users might expect images of music concerts, whereas other users might look for images of outdoor climbing. Ideally, a vision system would be aware of the multitude of different aspects. Figure 1.1 illustrates an interaction with such a potential vision system.

In this dissertation, I present approaches to address this challenge and build computer vision systems that can model contradicting points of view and adapt to different conditions. In our studies, we look at context-adaptive models from

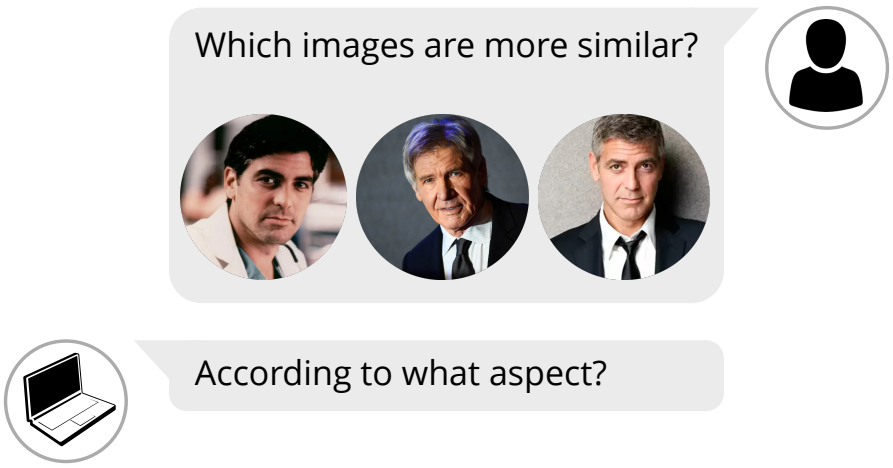


Figure 1.1: Example illustrating how images can be compared according to a multitude of semantic aspects. A successful vision system needs to take the visual concept into account that objects are compared to.

two different perspectives. First, we study models that can incorporate context information to personalize and customize results in the tasks of image classification and image similarity search. Second, we focus on the underlying convolutional neural network (Convnet) in a study that provides a better understanding of Convnets and further leads to the design of an architecture that can understand the context in which it is used and define its network topology adaptively, conditioned on the input image.

In Chapter 2, we develop conditional models in the context of image categorization. Here, the general goal is to learn a function that assigns images to a set of discrete categories or labels. This is one of the fundamental tasks for computer vision and important for a wide range of applications from image tagging to autonomous driving. The standard approach for this task comprises two main parts. A Convnet is trained to transform input images into a vector space, where each image can be represented by a vector, which is often called embedding vector. Further, one additional vector is learned for each category.

Images are then classified by choosing the categories with vectors closest to the image vector.

While effective on standard benchmark datasets, this model is inflexible in that the mapping between images and categories is fixed. To address this shortcoming, one could imagine to train a separate network for each user to account for different meanings that users might associate with images and labels. However, the idea is wasteful in terms of parameters and becomes infeasible by the associated need for training data. In our work, we present an alternative approach. Instead of only learning a mapping between images and labels, we jointly model images, labels and users. In particular, we learn vector representations for all three modalities and generate user-conditioned mappings between images and labels by conditioning the parameters of the mappings on the user embedding vector. Since the mapping is bidirectional, our approach allows to personalize both tag-based image retrieval as well as image tagging.

In Chapter 3, we study conditional perception in the area of visual similarity search. The goal here is to retrieve the most similar images to a given input image. Understanding image similarity provides the basis for many applications such as recommendation systems and visual search. Here, the typical approach is again to train a Convnet to transform images into a vector space representation. However, instead of jointly embedding images with a fixed set of categories, the network is trained such that the distances between the images' embedding vectors represent their relative semantic dis-similarity. One can then find related images to an input image by searching for nearest neighbors in the vector space.

This approach has a key limitation in that it relies on a single distance func-

tion between images, while images can be compared according to a multitude of semantic aspects. An illustrative example is the comparison of colored geometric shapes, a task toddlers are regularly exposed to with benefits to concept learning. Consider, that a red triangle and a red circle are very similar in terms of color, more so than a red triangle and a blue triangle. However, the triangles are more similar to one another in terms of shape than the triangle and the circle. To address this shortcoming, we introduce Conditional Similarity Networks, which can learn embeddings that gracefully deal with multiple notions of similarity. In particular, the embeddings are separated into multiple semantic subspaces such that different dimensions encode features for different notions of similarity. Then, by selecting the appropriate subset of dimensions, images can be compared to a specific notion of similarity, for example, according to color or shape. In the proposed approach, the Convnet that learns the embedding and the masks that learn to select relevant dimensions are trained jointly.

In Chapter 4, we point our focus towards the underlying Convnets themselves. Deep Convnets have enabled a series of major advances for many computer vision problems. Starting with image classification, they are now used for many other tasks as well, including object detection and segmentation. On a conceptual level, Convnets can be considered as a set of nonlinear transformations, called layers, that are sequentially applied to an input image. The parameters of the layers are trained so as to optimize the performance on a final task such as image classification. In recent years we have seen that network depth, i.e., the number of layers, plays a key role for classification performance. The seminal work on Residual Networks (Resnets) further showed that adding identity skip-connections bypassing each layer greatly improves optimization, allowing the training of much deeper networks.

The drastic increase in the number of layers raises the questions what functions the individual layers learn and whether every single layer is necessary. To shed light on this question, we perform a lesion study investigating the effects of the skip-connection. In our study, we find that although all layers in a Resnet are trained jointly, they exhibit a high degree of independence. In fact, almost any individual layer can be removed from a trained Resnet without interfering with other layers. Further, we find that the skip-connections ease training by introducing short paths that carry gradient throughout the extent of very deep networks. However, the short paths also lead to parameter redundancy as many layers learn very similar functions. These results demonstrate that convolutional networks don't need a fixed feed-forward structure and lead to the following research question: Could we assemble network graphs on the fly, conditioned on the input?

In Chapter 5, we build upon the result from the previous chapter and focus on how to design an adaptive Convnet architecture. Since layers in Resnets exhibit a high degree of independence, what if, after identifying the high-level concept of an image, a network could move directly to a layer that can distinguish fine-grained differences? Currently, a network would first need to execute sometimes hundreds of intermediate layers that specialize in unrelated aspects. Ideally, the more a network already knows about an image, the better it should be at deciding which layer to compute next.

To understand this goal, it is important to note that due to their success, Convnets are used to classify increasingly large sets of visually diverse categories. Thus, most parameters are used to model high-level features, because in contrast to low-level and many mid-level concepts, high-level features are not

broadly shared across categories. This means that for any given input image the number of computed features focusing on unrelated concepts increases. To address this challenge, we propose Adanets, convolutional networks that adaptively define their network topology conditioned on the input image. Following a high-level structure similar to Resnets, Adanets learn a set of convolutional layers and decide for each input image on the fly which layers are needed. By learning both general layers useful to all images and expert layers specializing on subsets of categories, Adanets allow only computing features relevant to the input image. It is worthy to note that Adanets do not require special supervision about label hierarchies and relationships. Our results demonstrate that Adanets both improve efficiency and also overall classification quality.

Chapter 6 concludes this dissertation and discusses future directions for this line of research.

CHAPTER 2

USER-CONDITIONAL IMAGE TAGGING AND RETRIEVAL

Convolutional networks have shown great success on image-classification tasks involving a small number of classes (1000s). An increasingly important question is how this success can be extended to tasks that require the recognition of a larger variety of visual content. An important obstacle to increasing variety is that successful recognition of the long tail of visual content [35] may require manual annotation of hundreds of millions of images into hundreds of thousands of classes, which is difficult and time-consuming.

Images annotated with hashtags provide an interesting alternative source of training data because: (1) they are available in great abundance, and (2) they describe the long tail of visual content that we would like to recognize. Furthermore, hashtags appear in the sweet spot between capturing much of the rich information contained in natural language descriptions [54] whilst being nearly as structured as image labels in datasets like ImageNet.

However, using hashtags as supervision comes with its own set of challenges. In addition to the missing-label problem that hampers many datasets with multi-label annotations (*e.g.*, [15, 49, 58]), hashtag supervision has the problem that *hashtags are inherently subjective*. Since hashtags are provided by users as a form of self-expression, some users may be using different hashtags to describe the same content (synonyms), whereas other users may be using the same hashtag to describe very different content (polysemy). As a result, hashtags cannot be treated as oracle descriptions of the visual content of an image, but must be viewed as user-dependent descriptions of that content.

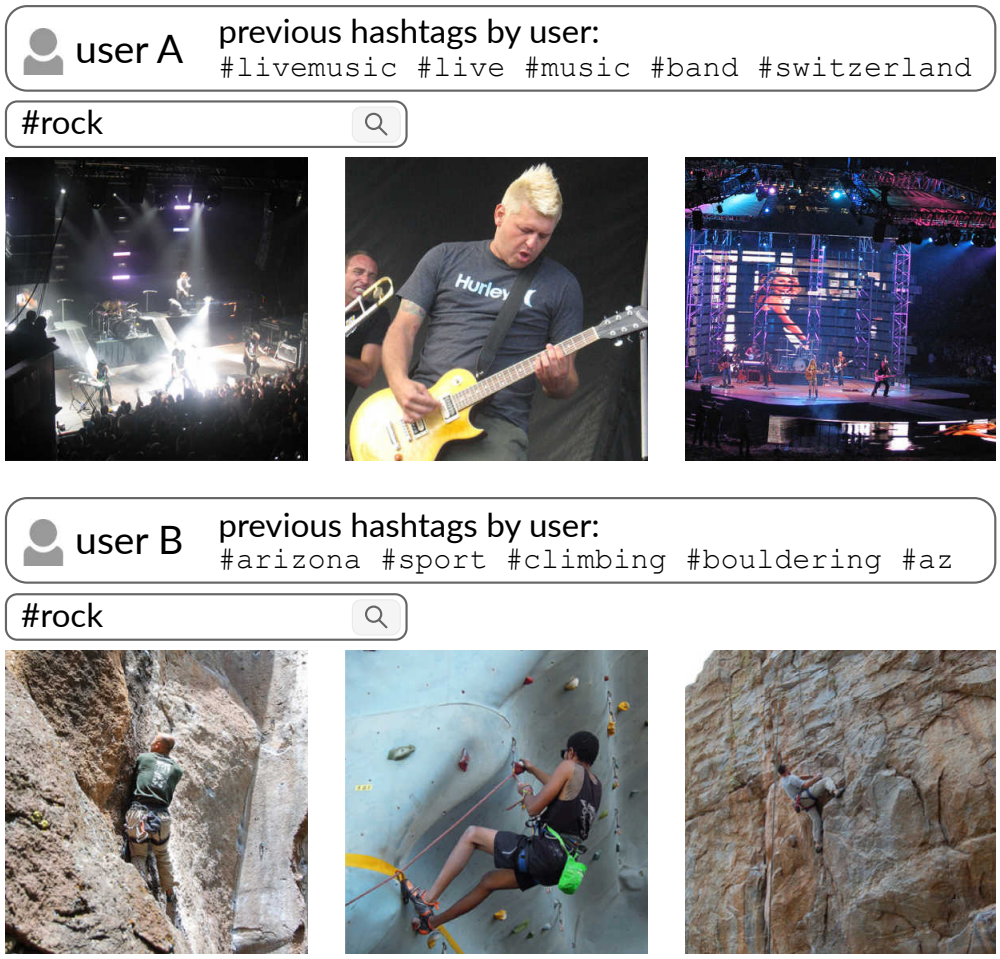


Figure 2.1: Image-retrieval results obtained using our user-specific hashtag model. The box above the query shows hashtags frequently used by the user in the past. Hashtag usage varies widely among users because they are a means of self-expression, not just a description of visual content. By jointly modeling users, hashtags, and images, our model disambiguates the query for a specific user. We refer the reader to the supplementary material for license information on the photos.

Motivated by this observation, we develop a *user-specific hashtag model* that takes the hashtag usage patterns of a user into account [102]. Instead of training on simple image-hashtag pairs, we train our model on image-user-hashtag triplets. This allows our model to learn patterns in the hashtag usage of a particular user, and to disambiguate the learning signal. After training, our model can perform a kind of intent determination that personalizes image retrieval

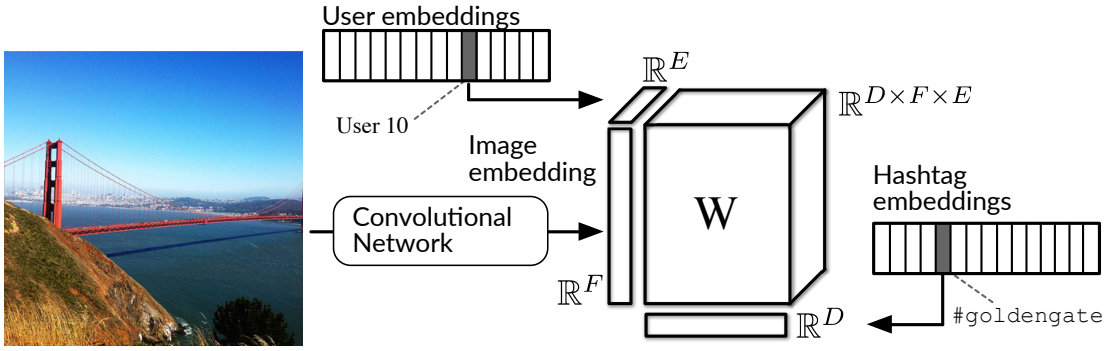


Figure 2.2: Overview of the proposed user-specific hashtag model. The three-way tensor product models the interactions between image features, hashtag embeddings, and user embeddings.

and tagging. This allows us to retrieve more relevant images and hashtags for a particular user. Figure 2.1 demonstrates how our user-specific hashtag model can disambiguate the ambiguous `#rock` hashtag by modeling the user.

Figure 2.2 provides an overview of this model. It is comprised of a convolutional network that feeds image features into a three-way tensor model, which is responsible for modeling the interaction between image features, hashtag embeddings, and an embedding that represents the user. When multiplying the three-way interaction tensor by a user embedding, we obtain a user-specific bilinear model. This personalized bilinear mapping between images and hashtags can take into account user-specific hashtag usage patterns. Our model can produce a single score for an image-hashtag-user triplet; we use this score in a ranking loss in order to learn parameters that discriminate between observed and unobserved triplets. The user embeddings are learned jointly with the weights of the three-way tensor model.

We investigate the efficacy of our models in (user-specific) image tagging and image retrieval experiments on the YFCC100M dataset [86]. We demon-

strate that: (1) we can learn to recognize large sets of visual concepts ranging from simple shapes to specific instances such as famous personalities and architectural landmarks by using hashtags as supervision; (2) our models successfully learn to discriminate synonyms and resolve hashtag ambiguities; and (3) we can improve accuracy on tasks such as image tagging by taking the user that uploaded the photo into account.

A preliminary version of this chapter has been published at CVPR 2018 [97].

2.1 Related Work

Our study is related to prior work on (1) hashtag prediction and recommendation, (2) large-scale weakly supervised training, and (3) three-way tensor models.

2.1.1 Hashtag Prediction and Recommendation

Several prior works have studied hashtag prediction and recommendation for text posts [24, 78], infographics [10], and images [18, 103]. The most closely related to our study is [18], which studies hashtag prediction conditioned on image and user features. The main differences between our work and [18] are (1) that we train the convolutional network end-to-end with hashtag supervision rather than pre-trained ImageNet features and (2) that the user embeddings in our model are *learned* based solely on the photos users posted and the hashtags they used. Our model does not receive any metadata about the user, whereas [18] assumes access to detailed user metadata. This allows us to model intent

on the level of individual users, which helps in disambiguating hashtags.

2.1.2 Large-scale Weakly Supervised Training

Our hashtag-prediction study is an example of large-scale weakly supervised training, which has been the topic of several recent studies. Specifically, [80] trains convolutional networks on 300 million images with noisy labels and show that the resulting models transfer to a range of other vision tasks. Similarly, [45, 54] train networks on the YFCC100M dataset to predict words or n-grams in user posts from image content, and explore transfer of these models to other vision tasks. Further, [93] explores augmenting large-scale weakly supervision with a small set of verified labels. Our study differs from these prior works both in terms of the type of supervision used (hashtags rather than manual annotation or n-grams from user comments), and in terms of its final objective (hashtag prediction rather than transfer to other vision problems).

2.1.3 Three-way Tensor Models

Tensor models have a long history in psychological data analysis [31, 87] and have increasingly been used in a wide range of machine-learning problems, including link prediction in relational and temporal graphs [20, 64], higher-order recommendation systems [67], and parameter estimation in latent variable models [2]. In computer vision, prominent examples of tensor models include the modeling of style and content [85], the joint analysis of image ensembles [92], sparse image coding [71] and gait recognition [25, 91].

2.2 Learning from Hashtags

Our goal is to train image-recognition models that can capture a large variety of visual concepts. In particular, we aim to learn from hashtags as supervisory signal. Formally, we assume access to a set of N images $\mathcal{I} = \{\mathbf{I}_1, \ldots, \mathbf{I}_N\}$ with height H, width W and C channels so that $\mathbf{I}_i \in [0,1]^{H \times W \times C}$, a vocabulary of K hashtags $\mathcal{H} = \{h_1, \ldots, h_K\}$, and a set of U users $\mathcal{U} = \{u_1, \ldots, u_U\}$. Each image is associated with a unique user, and with one or more hashtags (we discard images without associated hashtags from the dataset). The resulting dataset comprises a set of N triplets $\overline{\mathcal{T}}$, in which each triplet contains an image $\mathbf{I} \in \mathcal{I}$, a user $u \in \mathcal{U}$, and a hashtag set $\overline{\mathcal{H}} \subseteq \mathcal{H}$. Formally, $\overline{\mathcal{T}} = \{(\mathbf{I}_1, u_{m(1)}, \overline{\mathcal{H}}_1), \ldots, (\mathbf{I}_N, u_{m(N)}, \overline{\mathcal{H}}_N)\}$, in which $m(n)$ maps from the image/triplet index n to the corresponding user index in $\{1, \ldots, U\}$.

Hashtag supervision differs from traditional image annotations in that it was not intended to objectively describe the image content, but merely to serve as a medium for self-expression by the user. This self-expression leads to user-specific variation in hashtag supervision that is independent of the image content. We first study convolutional networks that are agnostic to the subjective nature of hashtags and simply treat them as image labels. Subsequently, we develop a user-specific model that explicitly incorporates the user as part of the hashtag-prediction model in order to capture variations in self-expression.

Throughout this work, we focus on two tasks: (1) a *tagging* task in which, given a query image $\mathbf{I}$, we aim to retrieve the most relevant hashtags for that image; and (2) a *retrieval* task in which, given a hashtag query $h \in \mathcal{H}$, we aim to retrieve the most relevant images for that hashtag.

2.2.1 User-Agnostic Hashtag Modeling

We investigate two approaches for training image-recognition models using user-agnostic hashtag supervision: (1) softmax multi-class classification and (2) hashtag-embedding regression [13]. In both cases, we learn an image model $f(\cdot;\theta) : [0,1]^{H\times W\times C} \rightarrow \mathbb{R}^D$ which maps images into an D-dimensional embedding space. The image model $f(\cdot;\theta)$ is implemented by a residual network [32] with parameters θ. In addition to the image model, we learn hashtag embeddings $\mathbf{h}_i \in \mathbb{R}^D$ for all hashtags $h_i \in \mathcal{H}$.

Multi-Class Classification. Several prior studies [45, 80] suggest that softmax classification can be very effective even in multi-label settings with large numbers of classes such as ours. Motivated by this, we train $f(\cdot;\theta)$ with a softmax over the $100,000$ most frequent hashtags by minimizing the multi-class logistic loss. Following [45], we select a single hashtag uniformly at random from hashtag set $\overline{\mathcal{H}}_n$ as target class for each image when training the softmax model. In particular, let $\mathbf{f}_j = f(\mathbf{I}_j;\theta) \in \mathbb{R}^D$ be the image embedding, and $h_i \in \overline{\mathcal{H}}_j$ the randomly selected hashtag. We then learn jointly the embeddings $\mathbf{h}_i$ and the parameters θ of the vision model $f(\cdot;\theta)$ by minimizing the negative log-likelihood for the probability distribution:

$$P(h_i|I_j) = \frac{\exp(\mathbf{h}_i^\top \mathbf{f}_j)}{\sum_\ell \exp(\mathbf{h}_\ell^\top \mathbf{f}_j)}. \tag{2.1}$$

Hashtag-Embedding Regression. This training method comprises two main stages. First, we learn an embedding $\mathbf{h}_i \in \mathbb{R}^D$ for each hashtag $h_i \in \mathcal{H}$. Second, we follow [13] and learn the parameters θ of the image model $f(\cdot;\theta)$ by minimizing the negative cosine similarity between the image embedding, $\mathbf{f}_j = f(\mathbf{I}_j;\theta) \in \mathbb{R}^D$, and the sum of the embeddings of the hashtags, $\overline{\mathbf{h}}_j$, corre-

sponding to image $\mathbf{I}_j$:

$$\ell(\mathbf{f}_j, \overline{\mathbf{h}}_j; \theta) = -\frac{\overline{\mathbf{h}}_j^\top \mathbf{f}_j}{||\overline{\mathbf{h}}_j|| ||\mathbf{f}_j||}. \tag{2.2}$$

A potential advantage of this approach is that the embeddings of synonomous hashtags are likely very similar: this implies that the loss used for training the convolutional network, in contrast to the multi-class logistic loss, does not substantially penalize predicting a synonymous hashtag that the user did not happen to use to describe the image.

We experiment with two methods for learning the hashtag embeddings $\mathbf{h}_i$. The first method computes the D principal singular vectors of the positive pointwise mutual information (PPMI) matrix [53]. The second method [56] explicitly models ambiguous hashtags (*i.e.*, hashtags with multiple meanings) by learning multi-sense hashtag embeddings. We follow [56] and use the global embedding vectors in their model as hashtag embedding. We train all models using mini-batch stochastic gradient descent (SGD).

2.2.2 User-Specific Hashtag Modeling

The models described above do not explicitly capture variations in hashtag labels that are due to variations in how users self-express. Here, we present a model that aims to capture these variations by modeling images, hashtags, and users jointly. We will show that this can help in disambiguating the meaning of hashtags assigned to images. As before, the model represents images via a convolutional network, $\mathbf{f}_j = f(\mathbf{I}_j; \theta) \in \mathbb{R}^F$, and hashtags via embeddings $\mathbf{h}_i \in \mathbb{R}^D$. In addition, we learn user embeddings, $\mathbf{u}_k \in \mathbb{R}^E$. We aim to learn a scoring function $s(t; \mathbf{W})$ with parameters $\mathbf{W} \in \mathbb{R}^{D \times F \times E}$ that combines all three representations

to predict whether or not an image-hashtag-user triplet t is correct. Specifically, we select a hashtag h_i from hashtag set $\overline{\mathcal{H}}_j$ uniformly at random, and model the score of the resulting triplet $t = (\mathbf{I}_j, u_k, h_i)$ as:

$$s(t; \mathbf{W}) = \sum_{r_1=1}^{D} \sum_{r_2=1}^{F} \sum_{r_3=1}^{E} w_{r_1 r_2 r_3} h_{i r_1} f_{j r_2} u_{k r_3}, \tag{2.3}$$

where $w_{r_1 r_2 r_3}$, $h_{i r_1}$, $f_{j r_2}$, and $u_{k r_3}$ are elements from $\mathbf{W}$, $\mathbf{h}_i$, $\mathbf{f}_j$, and $\mathbf{u}_k$, respectively. Equation 2.3 is a three-way tensor product between the embeddings of the image, hashtag, and user in which the weights $w_{r_1 r_2 r_3}$ specify the (positive or negative) interactions of all possible feature combinations. The user-specific aspect of Equation 2.3 can be observed by considering the summation over the user dimension. In particular, when summing over the user dimension, weighted by the embedding for user u_k, we obtain a user-specific weight matrix $\mathbf{W}^{(k)} \in \mathbb{R}^{D \times F}$ with entries:

$$w_{ab}^{(k)} = \sum_{r=1}^{E} u_{kr} w_{abr}. \tag{2.4}$$

The score function of Equation 2.3 is then equivalent to:

$$s(t; \mathbf{W}) = \mathbf{h}_i^\top \mathbf{W}^{(k)} \mathbf{f}_j. \tag{2.5}$$

Hence, our proposed model learns user-conditioned bilinear models between hashtags for images, by conditioning the weight matrix of the bilinear model on the user embedding.

Given a dataset of M triplets[1] $\mathcal{T} = \{t_1, \ldots, t_M\}$, we estimate the parameters $\mathbf{W}$ using a ranking approach. In particular, we want the score of a true observed triplet $t^+ \in \mathcal{T}$ to be higher than that of an unobserved triplet $t^- \notin \mathcal{T}$. We achieve this by minimizing the following loss:

$$\ell(t^+; \mathbf{W}) = \max\left(0, \max_{t^- \notin \mathcal{T}} s(t^-; \mathbf{W}) - s(t^+; \mathbf{W}) + 1)\right).$$

[1]Please note that $\mathcal{T}$ contains image-*hashtag*-user triplets, whereas $\overline{\mathcal{T}}$ contains image-*hashtag set*-user triplets.

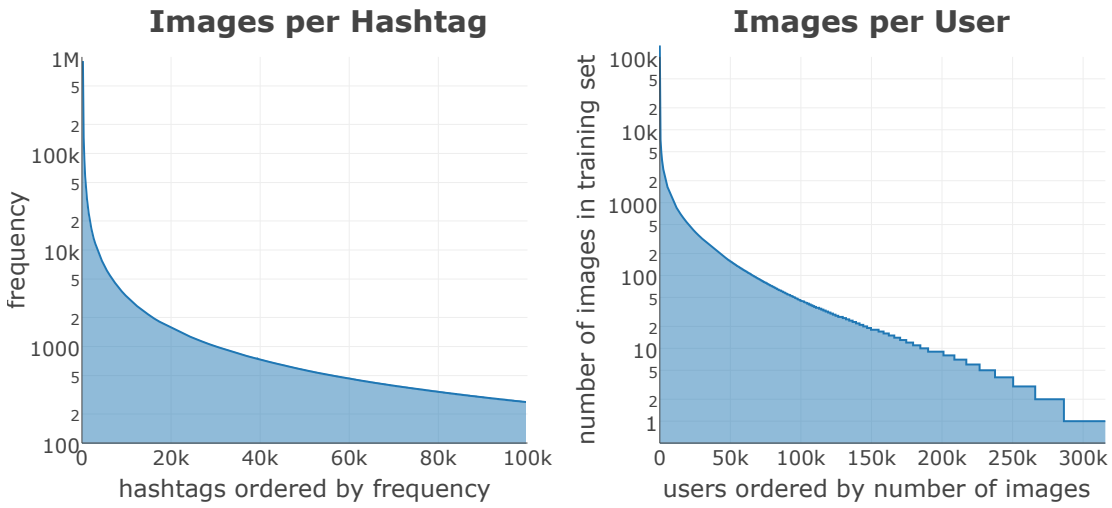


Figure 2.3: **Left:** Frequency of hashtags in hashtag vocabulary $\mathcal{H}$. **Right:** Number of photos per user in user set $\mathcal{U}$.

This ranking loss is better suited for our problem than a per-triplet binary logistic loss, because the latter would consider any unobserved triplet as a "negative". This is problematic because (1) the hashtag annotations for an image are generally not exhaustive and (2) there are far more unobserved than observed triplets. The ranking loss only aims to assign a lower score to unobserved triplets, and as a result it is not nearly as much affected by these problems.

In practice, the maximization over negative triplets t^- can only be approximated. For our ranking loss to be effective, it is essential to develop good approximations for the maximization by mining "hard negatives" [73]. We perform online hard negative mining along all three axes, *i.e.*, we rank tags, images, and users. Specifically, we sample six negative triplets per positive sample, and uses each of them as a negative in the loss. We sample three "intermediate" and three "hard" negatives. In an "intermediate" negative, one of the three elements (the image, hashtag, or user) of the positive triplet is replaced by another element that is selected uniformly at random from the training batch; the other

two elements remain the same. In a "hard" negative, we replace one of the three elements in the triplet by the (non-identical) element in the training batch that maximizes the score $s(t; \mathbf{W})$.

As before, we train our user-specific hashtag model using mini-batch SGD. We first learn the parameters of the convolutional network, θ, by minimizing one of the losses from 2.2.1. We then jointly learn the image, hashtag and user embeddings as well as the parameters of the scoring function, $\mathbf{W}$, in a subsequent training stage. In our experiments, we use image and hashtag embeddings with $D = F = 300$ dimensions and user embeddings of size $E = 50$.

Once we have inferred the embeddings for users, hashtags, and images as well as $\mathbf{W}$, we can then approach the aforementioned image tagging and retrieval results in the following way. Given a user u_k and an image $\mathbf{I}_j$, we compute the most likely hashtag according to our model as:

$$\arg\max_{h_i \in \mathcal{H}} \mathbf{h}_i^\top \mathbf{W}^{(k)} \mathbf{f}_j \tag{2.6}$$

The most likely image given a hashtag-user pair can be retrieved analogously.

2.3 Experiments

The aim of our experiments is: (1) to compare the strategies for training *user-agnostic* convolutional networks using hashtag supervision introduced in Section 2.2.1 and (2) to investigate the effectiveness of the *user-specific* hashtag model we introduced in Section 2.2.2.

Table 2.1: Frequency of the most common hashtags in the data set.

Hashtag	Frequency	Unique Users
#california	905,715	15,785
#travel	826,366	15,944
#usa	825,641	13,400
#london	764,277	21,516
#japan	732,859	11,652
#france	650,436	17,265
#wedding	580,605	19,599
#music	552,645	23,359
#beach	547,038	44,695

2.3.1 Dataset

We conduct experiments on the YFCC100M dataset [86] of approximately 99.2 million photos. More than 60 million of these photos have one or more associated hashtags, and each photo has an associated user, the user who uploaded it. We start by removing numerical hashtags and also remove the 10 most frequent tags because they are non-visual and non-informative (*e.g.*, #iphonography, #instagram, #square, and #canon). We define the hashtag set $\mathcal{H}$ as the set of the $100,000$ most frequent (remaining) hashtags. The left plot in Figure 2.3 shows the resulting hashtag frequencies, and Table 2.1 lists the most frequent hashtags. The hashtag distribution is heavily skewed towards a few frequent hashtags and has a long tail of less frequent tags. For example, the most frequent hashtag, #california, appears over $900,000$ times in the training set, *i.e.*, with 1.78% of training images. The least frequent hashtags in our hashtag set $\mathcal{H}$ only appear 260 times. Another characteristic of the hashtags is that while the most frequent tags tend to be English, less frequent tags are increasingly multilingual.

We select all photos with at least one hashtag from $\mathcal{H}$ and filter out photos by

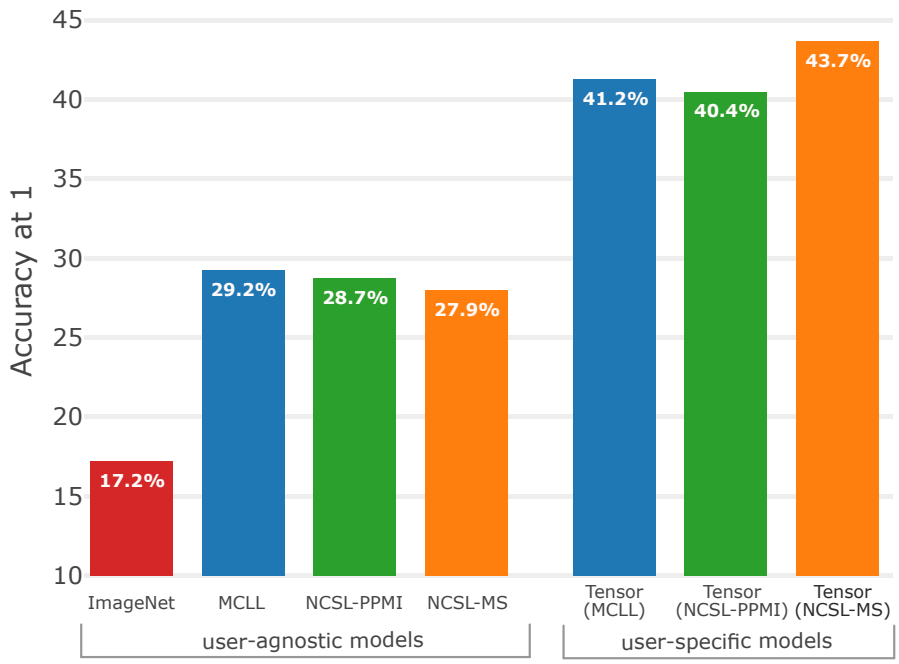


Figure 2.4: **Image tagging:** Accuracy@1 of four user-agnostic and three user-specific hashtag prediction models on the YFCC100M test set; see text for details. Higher is better.

"spammers", *i.e.*, by users that use more than 15 hashtags per image on average. This results in a dataset of 55.6 million images and a user set $\mathcal{U}$ with $U = 315,745$ users. As shown in the right plot in Figure 2.3, the number of photos per user is also heavy-tailed.

To model a realistic use-case, we split the photos for training and testing according to their upload time stamps. We sort the photos of each user by timestamp, assign the first 90% of the images to the training set, and assign the remaining photos to the validation and test sets. This results in a training set of $N = 50.6$ million photos, a validation set of 1 million images, and a test set of 4 million images. Taken together, the dataset contains 265 million hashtags for an average of about 4.7 tags per photo.

2.3.2 Hashtag Prediction

In the first set of experiments, we use our models to predict hashtags that are relevant to a given image. We measure the tagging quality of our models by their ability to predict the hashtags associated with the image in terms of accuracy@k ($A@k$). We denote the set of the k highest-scoring hashtags for image $\mathbf{I}_n$ by $\mathcal{R}_k^{(\mathbf{I}_n)}$, and as before, denote the set of hashtags that are associated to that image by $\overline{\mathcal{H}}_n$. Accuracy@k is then defined as:

$$A@k = \frac{1}{N}\sum_{n=1}^{N}\frac{\mathbb{I}\left[\mathcal{R}_k^{(\mathbf{I})_n} \cap \overline{\mathcal{H}}_n \neq \emptyset\right]}{N}. \tag{2.7}$$

We evaluate accuracy at $k=1$ and $k=10$ to measure (1) how often the top-ranked hashtag is in the ground-truth hashtag set and (2) how often at least one of the ground-truth hashtags appears in the 10 highest-ranked predictions. A key challenge in this task is that different users assign different hashtags to similar visual content: ideally, tagging methods assign hashtags that are relevant to the image content *and* are of importance to the user under consideration.

In addition to the user-specific model of Section 2.2.2, we evaluate four user-agnostic models: (1) a baseline model that trains a linear logistic regressor on features extracted by an convolutional network trained on ImageNet (**ImageNet**); (2) a network that is trained end-to-end for hashtag prediction using multi-class logistic loss (**MCLL**); (3) an end-to-end trained network that uses PPMI hashtag embeddings [53] in the negative cosine similarity loss of Equation 2.2 (**NCSL-PPMI**); and (4) an end-to-end trained network that uses the same loss but employs multi-sense hashtag embeddings [56] (**NCSL-MS**). In all experiments, our convolutional network is a ResNet-50. We evaluate three user-specific models that share the same architecture and training approach, but that vary in terms of the convolutional network that feeds image features into

Table 2.2: **Image tagging:** Accuracy@1 (A@1) and accuracy@10 (A@10) of two frequence baselines, four user-agnostic hashtag prediction models, and six user-specific hashtag prediction models; see text for details. Higher is better.

	Method	A@1	A@10
	Global frequency	1.68%	9.65%
	User-specific frequency	38.07%	62.55%
user-agnostic	Imagenet	17.21%	40.01%
user-agnostic	MCLL	29.24%	56.47%
user-agnostic	NCSL-PPMI	28.72%	47.70%
user-agnostic	NCSL-MS	27.94%	46.65%
user-specific	MLP (MCLL)	35.58%	65.58%
user-specific	MLP (NCSL-PPMI)	37.31%	67.68%
user-specific	MLP (NCSL-MS)	41.66%	71.34%
user-specific	Tensor (MCLL)	41.24%	70.75%
user-specific	Tensor (NCSL-PPMI)	40.43%	68.86%
user-specific	Tensor (NCSL-MS)	**43.65%**	**72.12%**

the three-way tensor model (those three networks were trained using MCLL, NCSL-PPMI, and NCSL-MS, respectively).

Figure 2.4 presents the tagging accuracy@1 of our four user-agnostic models three user-specific models on the test set. Additionally, Table 2.2 presents the accuracy@10 of these models, and three additional baselines: (1) a frequency baseline that predicts tags according to their frequency in the training set; (2) a *user-specific* frequency baseline that predicts tags according to their frequency for the user under consideration; and (3) a series of *user-specific* models in which we concatenate the embeddings of the three modalities and score them using a multi-layer perceptron (**MLP**) rather than the three-way tensor model.

From the results presented in the figure and the table, we make five main observations. First, all models clearly outperform the (global) frequency baseline and generally perform quite well given that each image can be assigned one of $100,000$ different hashtags. Second, the results show that training net-

works from scratch for hashtag prediction substantially outperforms Imagenet-trained networks, suggesting that the visual variety in ImageNet does not suffice for hashtag prediction. Third, the user-agnostic model that was trained using multi-class logistic loss (MCLL) outperforms user-agnostic trained using negative cosine similarity loss (NCSL), in particular, in terms of accuracy at 10. Fourth, all user-specific models significantly outperform the user-agnostic models, which demonstrates the ability of these models to capture user-specific features in their predictions. Fifth, the three-way tensor models substantially outperform the user-specific frequency baseline and generally outperform the user-specific MLP baselines models, which suggests three-way tensor models are best suited for tailoring predictions based on visual content to a particular user. The highest accuracy is obtained by a three-way tensor model on top of a convolutional network trained using NCSL-MS, which is surprising because that network has the lowest accuracy of the user-agnostic models.

In Figure 2.5, we break down the tagging accuracy of our models per user by measuring accuracy as a function of the number of training images the models observed for that user. We show the accuracy break-down for the best performing user-agnostic model (MCLL) and its corresponding tensor model. The figure shows that the user-agnostic model works well across all users, but tends to perform better for users with large image libraries. We surmise this effect is due to the fact that those users have provided the majority of the images in our training set, as a result of which they dominate the data distribution. For the user-specific tagging model, we observe a stronger relationship between accuracy and the number of images per user. Whilst the user-specific model outperforms the user-agnostic one for all users, the main benefits of the user-specific modeling are for users with more than approximately 27 uploaded photos. For

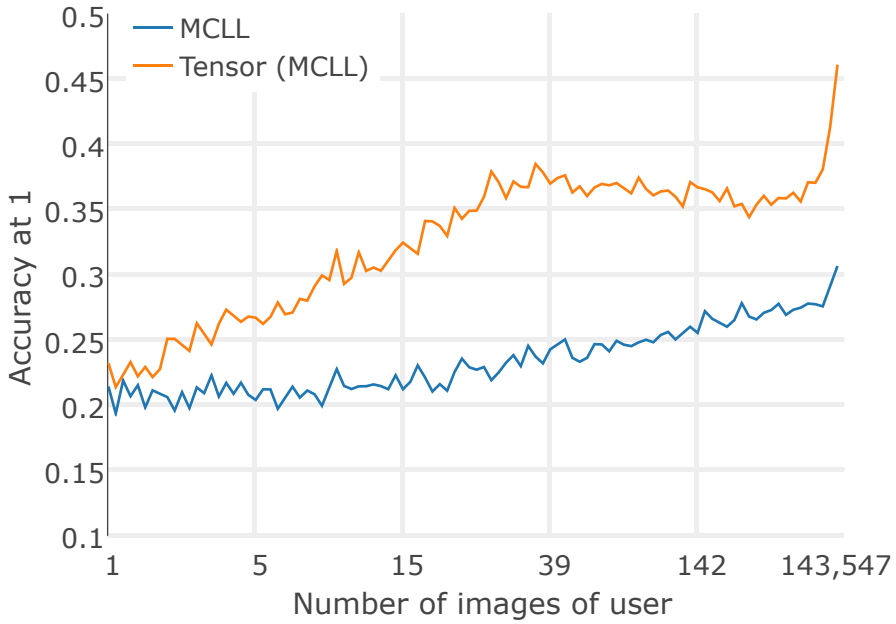


Figure 2.5: **Image tagging:** Accuracy@1 ($A@1$) of user-agnostic and respective user-specific tensor model as a function of the number of images by the user.

users with many photos, the tensor model has more data that it can use to pin down the user embeddings that capture their hashtag usage patterns.

Figure 2.6 shows examples of user-agnostic and user-specific tagging results. The tag predictions were obtained using the MCLL model and the Tensor (MCLL) model, respectively. The figure highlights the wide range of visual concepts that our convolutional networks learned to recognize. This range encompasses objects such as "people", "river", and "trees"; specific instances and locations such as the "Golden Gate Bridge", "San Francisco", and "New York"; whole-image concepts such as "autumn"; and image styles such as "black and white". The bottom part of the figure highlights the differences between the user-agnostic and user-specific models. Specifically, it shows tag predictions the user-agnostic model makes for a photo and predictions the user-specific model makes for that same photo *for a particular user* — we provide insight into the

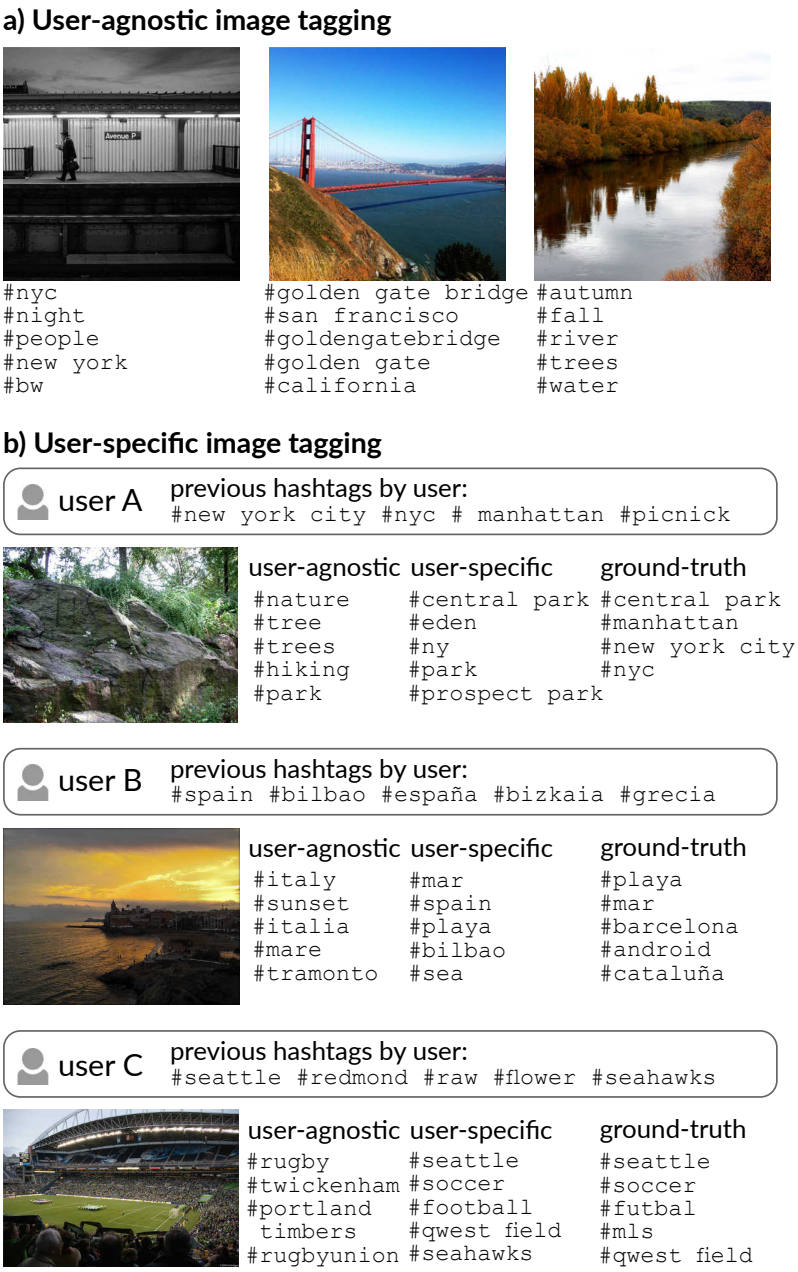


Figure 2.6: **Image tagging:** Example tagging results from the user-agnostic (MCLL) and user-specific (Tensor NCSL-MS) models.

user's "profile" by showing the most frequent hashtags for that user.

We observe that the user-specific model can help in disambiguating (most likely) locations of a photo: *e.g.,* it changes its prediction from `#nature` to

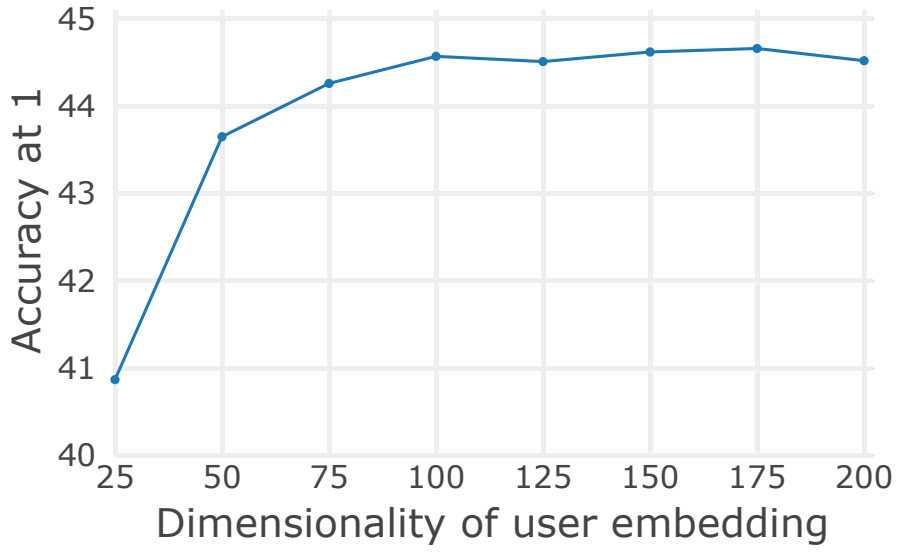


Figure 2.7: **Image tagging:** Accuracy@1 ($A@1$) of the Tensor (NCSL-MS) model as a function of the user embedding size, E.

`#central park` for a user that often tags photos with concepts related to New York. The user-specific model also can change predictions into the user's preferred language (*e.g.*, from English to Spanish), and it can help in disambiguating fine-grained categories, such as recognizing the difference between a rugby and a soccer stadium. We emphasize that all the information the user-specific model used to make these disambiguations comes from image-hashtag-user triples; the model does not employ any additional user metadata.

A key to the user-specific model are the user embeddings that personalize the mapping between images and hashtags. Figure 2.7 shows the accuracy of the top-performing user-specific model (Tensor NCSL-MS) as a function of the dimensionality of the user embedding. The results show that a substantial number of dimensions is needed, suggesting that the user embeddings are playing an important role in the accuracy of the model.

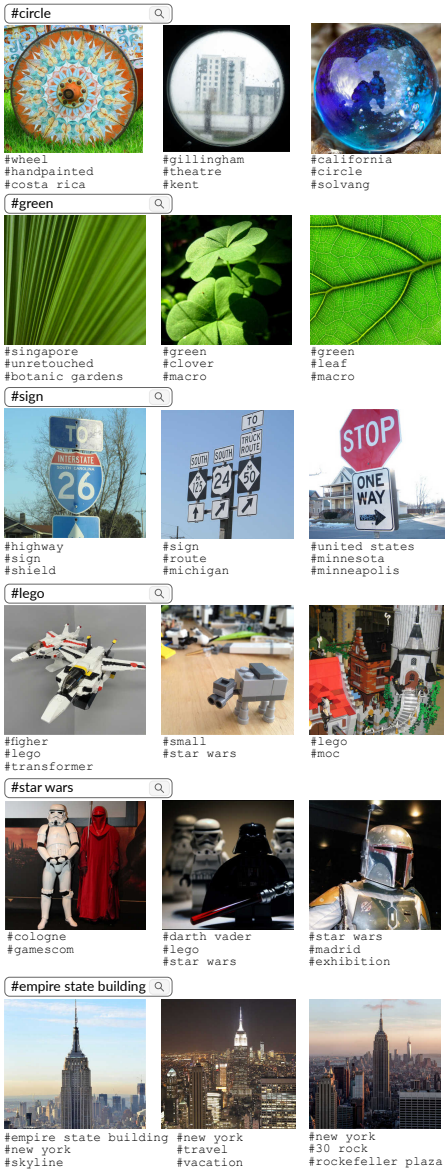


Figure 2.8: **Hashtag-based image retrieval:** Top-scoring photos and corresponding ground-truth hashtags for six hashtag queries. Results obtained using the user-agnostic MCLL model.

2.3.3 Hashtag-Based Image Retrieval

In a second set of experiments, we study hashtag-based image retrieval and measure the quality of our models by their ability to retrieve relevant images given a hashtag query in terms of precision@k ($P@k$). We define the set of the k

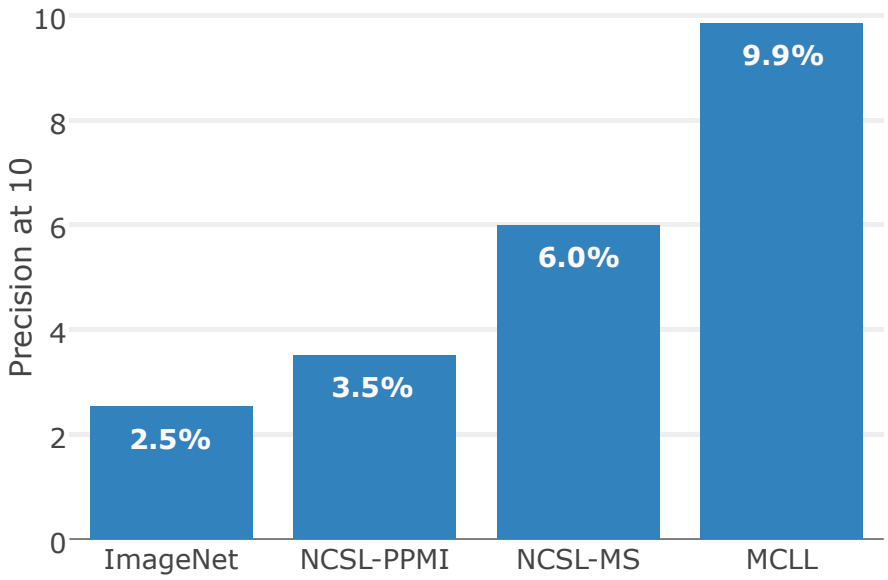


Figure 2.9: **Hashtag-based image retrieval:** Precision@10 ($P@10$) of four convolutional networks; see text for details. Higher is better.

highest-scoring images for hashtag h, $\mathcal{R}_k^{(h)}$, and the set of photos that are labeled with hashtag h, $\mathcal{GT}^{(h)} = \{\mathbf{I} \mid \exists u\colon (\mathbf{I}, u, h) \in \mathcal{T}\}$. Precision@$k$ is then defined as:

$$P@k = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} \frac{|\mathcal{R}_k^{(h)} \cap \mathcal{GT}^{(h)}|}{k}. \tag{2.8}$$

We measure $P@10$ in our experiments, *i.e.*, the fraction of the 10 top-scoring images that have the query hashtag associated with it. A key challenge in this task is that hashtags can have multiple meanings: ideally, retrieval methods retrieve photos corresponding to all meanings of a hashtag.

Figure 2.9 presents the $P@10$ on the test set for the four user-agnostic models that were also used in Section 2.3.2. From the results, we make three main observations. First, similar to the first experiment, the visual variety in ImageNet does not suffice for hashtag-based image retrieval, as reflected in the low precision of the ImageNet model. Second, multi-sense embeddings (MS) seem more suitable for training with the negative cosine similarity loss (NCSL) than PPMI embeddings, presumably, because they are better at modeling ambiguous

hashtags. Third, we observe that the network that was trained using multi-class logistic loss (MCLL) substantially outperforms all other models.

We emphasize that not every relevant photo for a hashtag query is also labeled with that hashtag, which gives rise to the relatively low precision values in Figure 2.9. We show qualitative image-retrieval results produced by the MCLL model in Figure 2.8, which suggest that many of the retrieved photos are actually relevant to the hashtag queries, even if they are not labeled as such. More importantly, Figure 2.8 illustrates the wide variety of visual concepts our models learned to recognize; the concepts recognized range from simple shapes and colors to fine-grained concepts and individual instances of architectural landmarks. Figure 2.1 shows an example of images retrieved by our user-specific model for the same query, `#rock`, for two different users. The figure demonstrates how modeling the user can help to disambiguate hashtag queries.

In Figure 2.10, we break down the image-retrieval precision by the frequency of the hashtags we query. The plot shows that: (1) retrieval performance is higher for frequent tags and (2) the difference between the MCLL model and the NCSL models is primarily in the long tail of less frequent tags. When evaluated on the $1,000$ most frequent tags, the classification and the multi-sense embedding model achieve a very similar precision@10 of 47%.

We surmise the relatively poor performance of the embedding-regression (NCSL) models in our image-retrieval experiments is due to the hashtag embeddings being fixed in those models, whereas they are learned jointly with the visual features in the classification model. This reduces the effective capacity of embedding-regression models, resulting in weaker performances. This limitation is alleviated in the user-specific model, in which all embeddings are

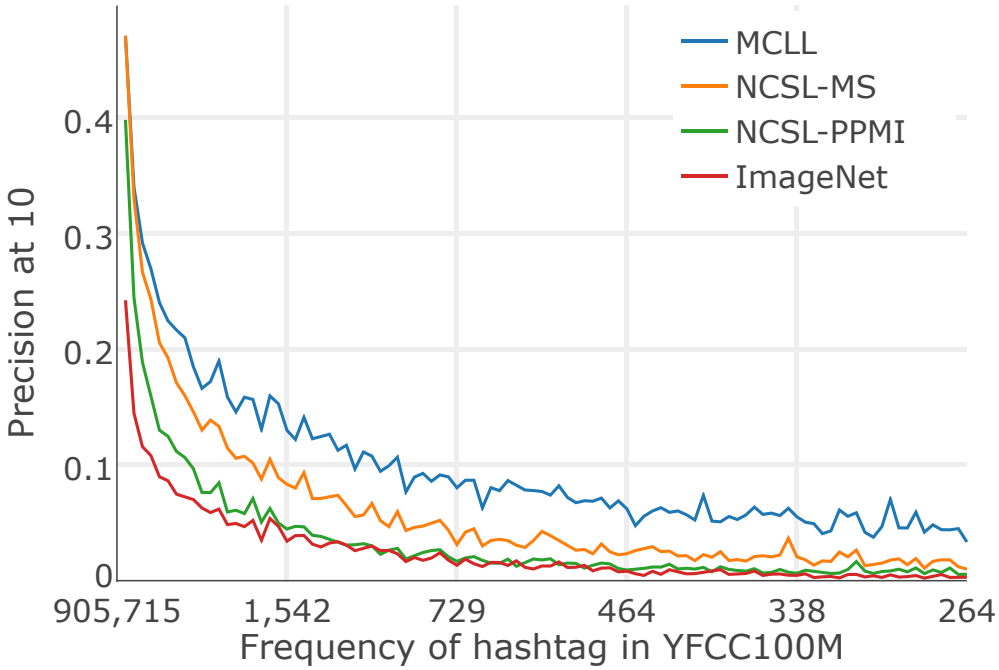


Figure 2.10: **Hashtag-based image retrieval:** Precision@10 ($P@10$) of four convolutional networks as a function of the frequency of the hashtag query. Higher is better.

learned jointly. For example, we observe competitive performance of the tensor model that builds on NCSL-trained convolutional networks in the tagging experiments.

2.4 Conclusion

This paper trained convolutional networks from scratch to perform hashtag prediction, and extended these networks with a three-way tensor model that learns user embeddings jointly with the final prediction model. This allows us to tailor the model's prediction to a specific user at test time. We used two different approaches for training the convolutional networks: a standard classification ap-

proach and an approach that regresses onto pre-learned hashtag embeddings. The classification approach performs consistently well across all tasks, whereas the embedding-regression approach mainly performs well for (user-specific) image tagging. Generally, the user-specific approach significantly outperforms the user-agnostic models demonstrating the ability to capture user-specific features in the predictions.

In future work, we intend to re-visit user-specific image retrieval in a setting in which explicit relevance information is available. Other directions for future work include incorporating user metadata [18] as well as spatial and temporal patterns [65] in our model.

CHAPTER 3

CONDITIONAL SIMILARITY NETWORKS

Understanding visual similarities between images is a key problem in computer vision. To measure the similarity between images, they are embedded in a feature-vector space, in which their distances preserve the relative dissimilarity. Commonly, convolutional neural networks are trained to transform images into respective feature-vectors. We refer to these as Similarity Networks. When learning such networks from pairwise or triplet (dis-)similarity constraints, the simplifying assumption is commonly made that objects are compared according to one unique measure of similarity. However, objects have various attributes and can be compared according to a multitude of semantic aspects.

An illustrative example to consider is the comparison of coloured geometric shapes, a task toddlers are regularly exposed to with benefits to concept learning. Consider, that a red triangle and a red circle are very similar in terms of color, more so than a red triangle and a blue triangle. However, the triangles are more similar to one another in terms of shape than the triangle and the circle.

An optimal embedding should minimize distances between perceptually similar objects. In the example above and also in the practical example in Figure 3.1 this creates a situation where the same two objects are semantically repelled and drawn to each other at the same time. A standard triplet embedding ignores the sources of similarity and cannot jointly satisfy the competing semantic aspects. Thus, a successful embedding necessarily needs to take the visual concept into account that objects are compared to.

One way to address this issue is to learn separate triplet networks for each

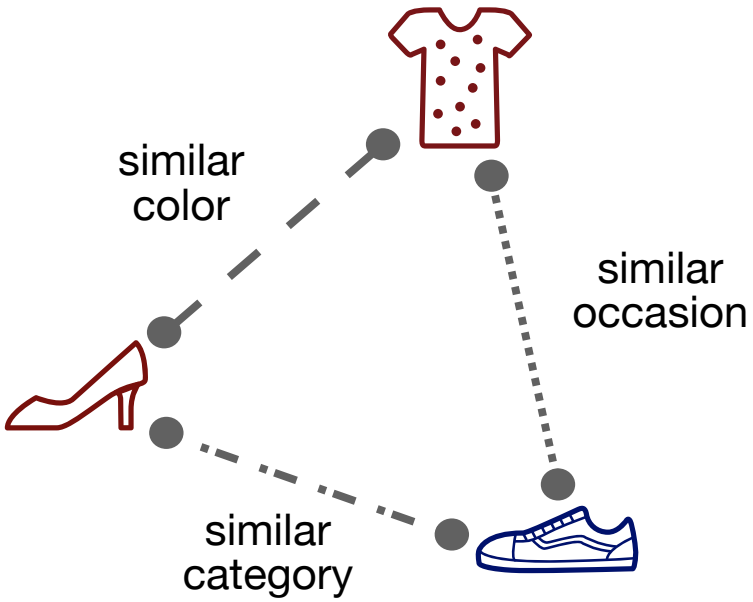


Figure 3.1: Example illustrating how objects can be compared according to multiple notions of similarity. Here, we demonstrate three intuitive concepts, which are challenging to combine for a machine vision algorithm that has to embed objects in a feature space where distances preserve the relative dissimilarity: shoes are of the same category; red objects are more similar in terms of color; sneakers and t-shirts are stylistically closer.

aspect of similarity. However, the idea is wasteful in terms of parameters needed, redundancy of parameters, as well as the associated need for training data.

In this work, we introduce Conditional Similarity Networks (CSNs) a joint architecture to learn a nonlinear embeddings that gracefully deals with multiple notions of similarity within a shared embedding using a shared feature extractor. Different aspects of similarity are incorporated by assigning responsibility weights to each embedding dimension with respect to each aspect of similarity. This can be achieved through a masking operation leading to separate semantic subspaces. Figure 3.2 provides an overview of the proposed framework. Images are passed through a convolutional network and projected into a nonlinear

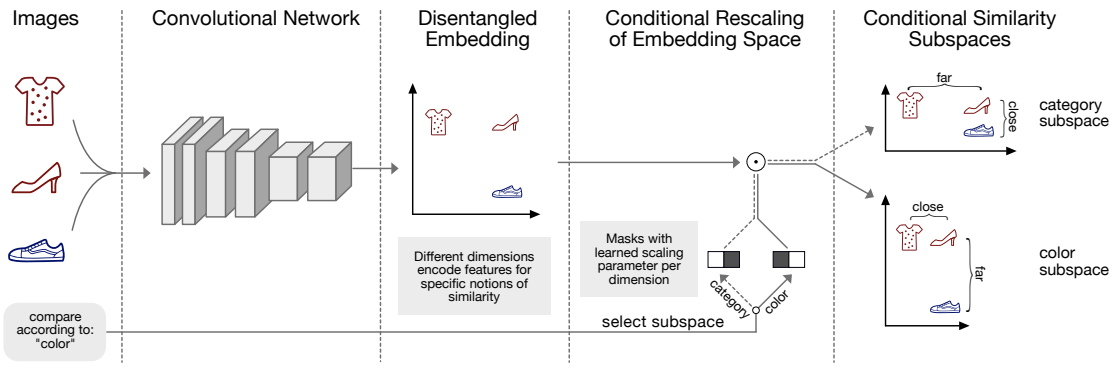


Figure 3.2: The proposed Conditional Similarity Network consists of three key components: First, a learned convolutional neural network as feature extractor that learns the disentangled embedding, i.e., different dimensions encode features for specific notions of similarity. Second, a condition that encodes according to which visual concept images should be compared. Third, a learned masking operation that, given the condition, selects the relevant embedding dimensions that induce a subspace which encodes the queried visual concept.

embedding such that different dimensions encode features for specific notions of similarity. Subsequent masks indicate which dimensions of the embedding are responsible for separate aspects of similarity. We can then compare objects according to various notions of similarity by selecting an appropriate masked subspace. In the proposed approach the convolutional network that learns the disentangled embedding as well as the masks that learn to select relevant dimensions are trained jointly.

In our experiments we evaluate the quality of the learned embeddings by their ability to embed unseen triplets. We demonstrate that CSNs clearly outperform single triplet networks, and even sets of specialist triplet networks where a lot more parameters are available and each network is trained towards one single similarity notion. Further we show CSNs make the representation interpretable by encoding different similarities in separate dimensions.

Our contributions are a) formulating Conditional Similarity Networks, an

approach that allows to to learn nonlinear embeddings that incorporate multiple aspects of similarity within a shared embedding using a shared feature extractor, b) demonstrating that the proposed approach outperforms standard triplet networks and even sets of specialist triplet networks in a variety of hard predictive visual tasks and c) demonstrating that our approach successfully disentangles the embedding features into meaningful dimensions so as to make the representation interpretable.

A preliminary version of this chapter has been published at CVPR 2017 [95].

3.1 Related Work

3.1.1 Similarity Based Learning

Similarity based learning has emerged as a broad field of interest in modern computer vision and has been used in many contexts. Disconnected from the input image, triplet based similarity embeddings, can be learned using crowd-kernels [90]. Further, Tamuz et al. [83] introduce a probabilistic treatment for triplets and learn an adaptive crowd kernel. Similar work has been generalized to multiple-views and clustering settings by Amid and Ukkonen [1] as well as Van der Maaten and Hinton [89]. A combination of triplet embeddings with input kernels was presented by Wilber et al. [104], but this work did not include joint feature and embedding learning. An early approach to connect input features with embeddings has been to learn image similarity functions through ranking [11].

3.1.2 Deep Metric Learning

A foundational line of work combining similarities with neural network models to learn visual features from similarities revolves around Siamese networks [14, 30], which use pairwise distances to learn embeddings discriminatively. Related to our work, pairwise similarities have been used to learn visual clothing style [96]. In contrast to pairwise comparisons, triplets have a key advantage due to their flexibility in capturing a variety of higher-order similarity constraints rather than the binary similar/dissimilar statement for pairs. Neural networks to learn visual features from triplet based similarities have been used by Wang et al. [100] and Schroff et al. [69] for face verification and fine-grained visual categorization. A key insight from these works is that semantics as captured by triplet embeddings are a natural way to represent complex class-structures when dealing with problems of high-dimensional categorization and greatly boost the ability of models to share information between classes.

3.1.3 Disentangling Representations

Disentangling representations is a major topic in the recent machine learning literature and has for example been tackled using Boltzmann Machines by Reed et al. [66]. Chen et al. [12] propose information theoretical factorizations to improve unsupervised adversarial networks. Within this stream of research, the work closest to ours is that of Karaletsos et al. [46] on representation learning which introduces a joint generative model over inputs and triplets to learn a factorized latent space. However, the focus of that work is the generative aspect of disentangling representations and proof of concept applications to low-

dimensional data. Our work introduces a convolutional embedding architecture that forgoes the generative pathway in favor of exploring applications to embed high-dimensional image data. We thus demonstrate that the generative interpretation is not required to reap the benefits of Conditional Similarity Networks and demonstrate in particular their use in common computer vision tasks.

3.1.4 Factorizing Latent Spaces

A theme in our work is the goal of modeling separate similarity measures within the same system by factorizing (or *disentangling*) latent spaces. We note the relation of these goals to a variety of approaches used in representation learning. Multi-view learning [79, 101] has been used for 3d shape inference and shown to generically be a good way to learn factorized latent spaces. Multiple kernel learning [6, 75] employs information encoded in different kernels to provide predictions using the synthesized complex feature space and has also been used for similarity-based learning by McFee and Lanckriet [62]. Multi-task learning approaches [16] are used when information from disparate sources or using differing assumptions can be combined beneficially for a final prediction task. Indeed, our gating mechanism can be interpreted as an architectural novelty in neural networks for *multi-task triplet learning*. Similar to our work, multiliniear networks [59] also strive to factorize representations, but differ in that they ignore weak additional information. An interesting link also exists to multiple similarity learning [5], where category specific similarities are used to approximate a fine-grained global embedding. Our global factorized embeddings can be thought of as an approach to capture similar information in a shared space

directly through feature learning.

3.1.5 Attention Mechanisms

We also discuss the notion of attention in our work, by employing gates to attend to the mentioned subspaces of the inferred embeddings when focusing on particular visual tasks. This term may be confused with spatial attention such as used in the DRAW model [27], but bears similarity insofar as it shows that the ability to gate the focus of the model on relevant dimensions (in our case in latent space rather than observed space) is beneficial both to the semantics and to the quantitative performance of our model.

3.2 Conditional Similarity Networks

Our goal is to learn a nonlinear feature embedding $f(x)$, from an image x into a feature space $\mathbb{R}^d$, such that for a pair of images x_1 and x_2, the Euclidean distance between $f(x_1)$ and $f(x_2)$ reflects their semantic dis-similarity. In particular, we strive for the distance between images of semantically similar objects to be small, and the distance between images of semantically different objects to be large. This relationship should hold independent of imaging conditions.

We consider $y = f(x)$ to be an embedding of observed images x into coordinates in a feature space y. Here, $f(x) = Wg(x)$ clarifies that the embedding function is a composition of an arbitrarily nonlinear function $g(\cdot)$ and a linear projection W, for $W \in \mathbb{R}^{d \times b}$, where d denotes the dimensions of the embedding and b stands for the dimensions of the output of the nonlinear function $g(\cdot)$. In

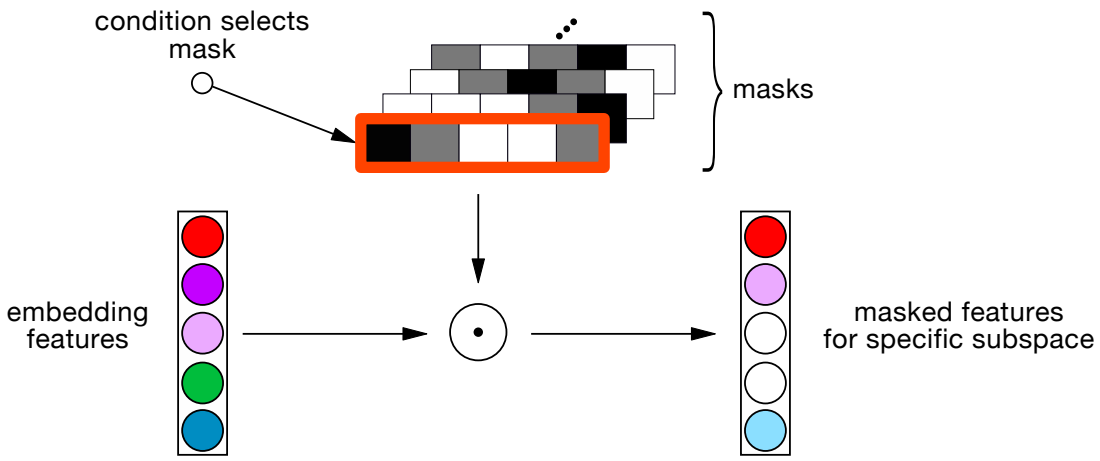


Figure 3.3: The masking operation selects relevant embedding dimensions, given a condition index. Masking can be seen as a soft gating function, to attend to a particular concept.

general, we denote the parameters of function $f(x)$ by θ, denoting all the filters and weights.

3.2.1 Conditional Similarity Triplets

Apart from observing images $\mathbf{x}$, we are also given a set of triplet constraints sampled from an oracle such as a crowd. We define triplet constraints in the following.

Given an unknown conditional similarity function $s_c(\cdot, \cdot)$, an oracle such as a crowd can compare images x_1, x_2 and x_3 according to condition c. A condition is defined as a certain notion of similarity according to which images can be compared. Figure 3.1 gives a few example notions according to which images of fashion products can be compared. The condition c serves as a switch between attented visual concepts and can effectively gate between different similarity functions s_c. Using image x_1 as reference, the oracle can

apply $s_c(x_1, x_2)$ and $s_c(x_1, x_3)$ and decide whether x_1 is more similar to x_2 or to x_3 conditioned on c. The oracle then returns an ordering over these two distances, which we call a triplet t. A triplet is defined as the set of indices {reference image, more distant image, closer image}, e.g. $\{1, 3, 2\}$ if $s_c(x_1, x_3)$ is larger than $s_c(x_1, x_2)$.

We define the set of all triplets related to condition C as:

$$\mathcal{T}_\rfloor = \{(i, j, l; c) \mid s_c(x_i, x_j) > s_c(x_i, x_l)\}. \tag{3.1}$$

We do not have access to the exhaustive set $\mathcal{T}_\rfloor$, but can sample K-times from it using the oracle to yield a finite sample $\mathcal{T}_\rfloor^K = \{t_k\}_{k=1}^K$.

3.2.2 Learning From Triplets

The feature space spanned by our model is given by function $f(\cdot)$. To learn this nonlinear embedding and to be consistent with the observed triplets, we define a loss function $L_T(\cdot)$ over triplets to model the similarity structure over images. The triplet loss commonly used is

$$L_T(x_i, x_j, x_l) = \max\{0, D(x_i, x_j) - D(x_i, x_l) + h\} \tag{3.2}$$

where $D(x_i, x_j) = \|f(x_i; \theta) - f(x_j; \theta)\|_2$. is the Euclidean distance between the representations of images x_i and x_j. The scalar margin h helps to prevent trivial solutions. The generic triplet loss is not capable of capturing the structure induced by multiple notions of similarities.

To be able to model conditional similarities, we introduce masks $\mathbf{m}$ over the embedding with $m \in \mathbb{R}^{d \times n_c}$ where n_c is the number of possible notions of similarities. We define a set of parameters β_m of the same dimension as $\mathbf{m}$ such that

$\mathbf{m} = \sigma(\beta)$, with σ denoting a rectified linear unit so that $\sigma(\beta) = \max\{0, \beta\}$. As such, we denote m_c to be the selection of the c-th mask column of dimension d (in pseudocode $m_c = \mathbf{m}[:, c]$). The mask plays the role of an element-wise gating function selecting the relevant dimensions of the embedding required to attend to a particular concept. The role of the masking operation is visually sketched in Figure 3.3. The masked distance function between two images x_i and x_j is given by

$$D(x_i, x_j; m_c, \theta) = \|f(x_i; \theta)m_c - f(x_j; \theta)m_c\|_2. \tag{3.3}$$

While appearing to be a small technical change, the inclusion of a masking mechanism for the triplet-loss has a highly non-trivial effect. The mask induces a subspace over the relevant embedding dimensions, effectively attending only to the relevant dimensions for the visual concept being queried. In the loss function above, that translates into a modulated cost phasing out Euclidean distances between irrelevant feature-dimensions while preserving the loss-structure of the relevant ones.

Given an triplet $t = \{i, j, l\}$ defined over indices of the observed images and a corresponding condition-index c, the final triplet loss function $\mathcal{L}_T(\cdot)$ is given by:

$$\begin{aligned} &\mathcal{L}_T(x_i, x_j, x_l, c; m, \theta) = \\ &\max\{0, D(x_i, x_j; m_c, \theta) - D(x_i, x_l; m_c, \theta) + h\} \end{aligned} \tag{3.4}$$

3.2.3 Encouraging Regular Embeddings

We want to encourage embeddings to be drawn from a unit ball to maintain regularity in the latent space. We encode this in an embedding loss function $\mathcal{L}_W$

given by:

$$\mathcal{L}_W(\mathbf{x};\theta) = \|f(\mathbf{x};\theta)\|_2^2 = \|\mathbf{y}\|_2^2 \tag{3.5}$$

The separate subspaces are computed as $f(x)m_c$. To prevent the masks from expanding the embedding and to encourage sparse masks, we add a loss to regulate the masks:

$$\mathcal{L}_M(\mathbf{m}) = \|\mathbf{m}\|_1 \tag{3.6}$$

Without these terms, an optimization scheme may choose to inflate embeddings to create space for new data points instead of learning appropriate parameters to encode the semantic structure.

3.2.4 Joint Formulation For Convolutional CSNs

We define a loss-function $\mathcal{L}_{CSN}$ for training CSNs by putting together the defined loss functions. Given images $\mathbf{x}$, triplet constraints with associated condition $\{\mathbf{t},\ \mathbf{c}\}$ as well as parameters for the masks $\mathbf{m}$ and the embedding function θ, the CSN loss is defined as

$$\begin{aligned} &\mathcal{L}_{CSN}(\mathbf{x}, \{\mathbf{t}, \mathbf{c}\}; \mathbf{m}, \theta) = \\ &\mathcal{L}_T(x_{t_0}, x_{t_1}, x_{t_2}, c; \mathbf{m}, \theta) + \lambda_1 \mathcal{L}_W(\mathbf{x}, \theta) + \lambda_2 \mathcal{L}_M(\mathbf{m}) \end{aligned} \tag{3.7}$$

The parameters λ_1 and λ_2 weight the contributions of the triplet terms against the regular embedding terms.

In our paper, the nonlinear embedding function is defined as $f(x) = Wg(x)$, where $g(x)$ is a convolutional neural network. In the masked learning procedure the masks learn to select specific dimensions in the embedding that are associated with a given notion of similarity. At the same time, $f(\cdot)$ learns to encode the

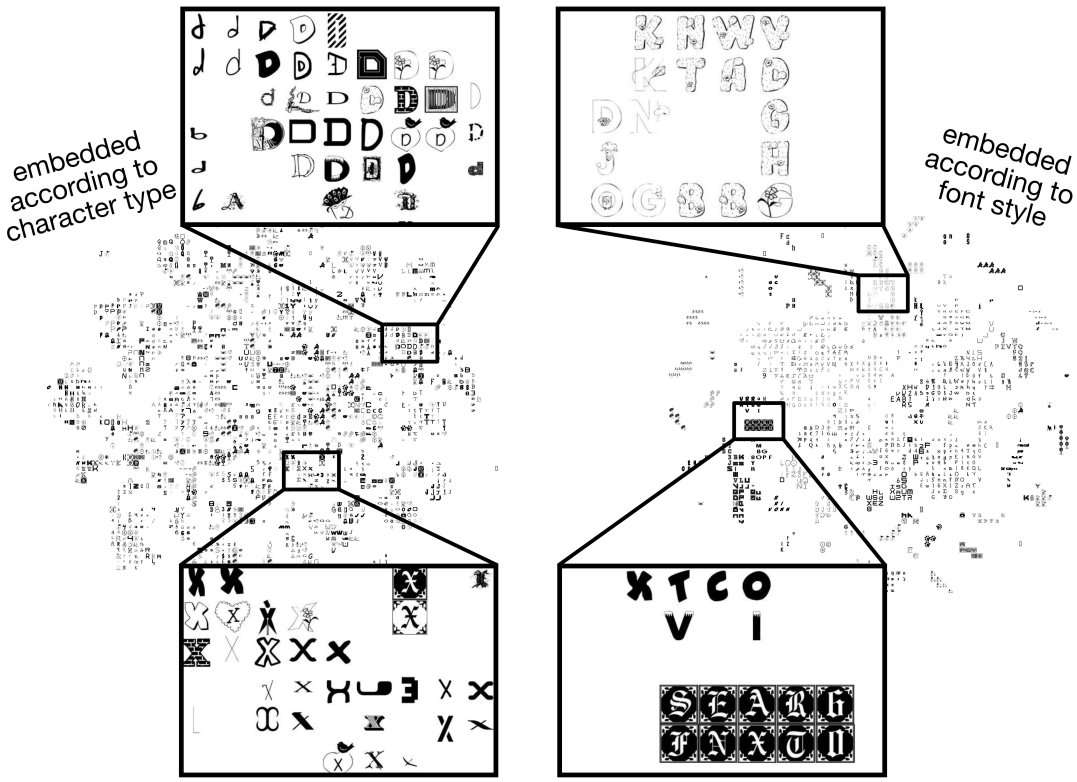


Figure 3.4: Visualization of 2D embeddings of two learned subspaces of the character feature space. The subspaces are obtained by attending to different subsets of the dimensions in the image representations. The subspace on the left groups images by character type, the one on the right according to font style. For clear visual representation we discretize the space into a grid and pick one image from each cell at random.

visual features such that different dimensions in the embedding encode features associated to specific semantic notions of similarity. Then, during test time each image can be mapped into this embedding by $f(\cdot)$. By looking at the different dimensions of the image's representation, one can reason about the different semantic notions of similarity. We call a feature space spanned by a function with this property *disentangled*, as it preserves the separation of the similarity notions through test time.

(a) Embedding according to the closure mechanism

(b) Embedding groups of boots, slippers, shoes and sandals

Figure 3.5: Visualization of 2D embeddings of subspaces learned by the CSN. The spaces are clearly organized according to **(a)** closure mechanism of the shoes and **(b)** the category of the shoes. This shows that CSNs can successfully separate the subspaces.

3.3 Experiments

We focus our experiments on evaluating the semantic structure of the learned embeddings and their subspaces as well as the underlying convolutional filters.

3.3.1 Datasets

We perform experiments on two different datasets. First, for illustrative purposes we use a dataset of fonts[1] collected by Bernhardsson. The dataset contains 3.1 million images of single characters in gray scale with a size of 64 by 64 pixels each. The dataset exhibits variations according to *font style* and *character type*. In particular, it contains 62 different characters in 50,000 fonts, from which we use the first 1,000. Second, we use the Zappos50k shoe dataset [107] collected by Yu and Grauman. The dataset contains 50,000 images of individual richly annotated shoes, with a size of 136 by 102 pixels each, which we resize to 112 by 112. The images exhibit multiple complex variations. In particular, we are looking into four different characteristics: the *type of the shoes* (i.e., shoes, boots, sandals or slippers), the *suggested gender of the shoes* (i.e., for women, men, girls or boys), the *height of the shoes' heels* (numerical measurements from 0 to 5 inches) and the *closing mechanism of the shoes* (buckle, pull on, slip on, hook and loop or laced up). We also use the shoes' brand information to perform a fine-grained classification test.

To supervise and evaluate the triplet networks, we sample triplet constraints from the annotations of the datasets. For the font dataset, we sample triplets such that two characters are of the same type or font and one is different. For the Zappos dataset, we sample triplets in an analogous way for the three categorical attributes. For the heel heights we have numerical measurements so that for each triplet we pick two shoes with similar height and one with different height. First, we split the images into three parts: 70% for training, 10% for validation and 20% in the test set. Then, we sample triplets within each set. For each

[1] http://erikbern.com/2016/01/21/analyzing-50k-fonts-using-deep-neural-networks/

attribute we collect 200k train, 20k validation and 40k test triplets.

3.3.2 Baselines and Model Variants

As initial model for our experiments we use a ConvNet pre-trained on ImageNet. All model variants are fine-tuned on the same set of triplets and only differ in the way they are trained. We compare four different approaches, which are schematically illustrated in Figure 3.6.

Standard Triplet Network: The common approach to learn from triplet constrains is a single Convolutional Network where the embedding layer receives supervision from the triplet loss defined in Equation 3.2. As such, it aims to learn from all available triplets jointly as if they come from a single measure of similarity.

Set of Task Specific Triplet Networks: Second, we compare to a set of n_c separate triplet network experts, each of which is trained on a single notion of similarity. This overcomes the simplifying assumption that all comparisons come from a single measure of similarity. However, this comes at the cost of significantly more parameters. This is the best model achievable with currently available methods.

Conditional Similarity Networks - fixed disjoint masks: We compare two variants of Conditional Similarity Networks. Both extend a standard triplet network with a masking operation on the embedding vector and supervise the network with the loss defined in Equation 3.4. The first variant learns the convolutional filters and the embedding. The masks are pre-defined to be disjoint between the

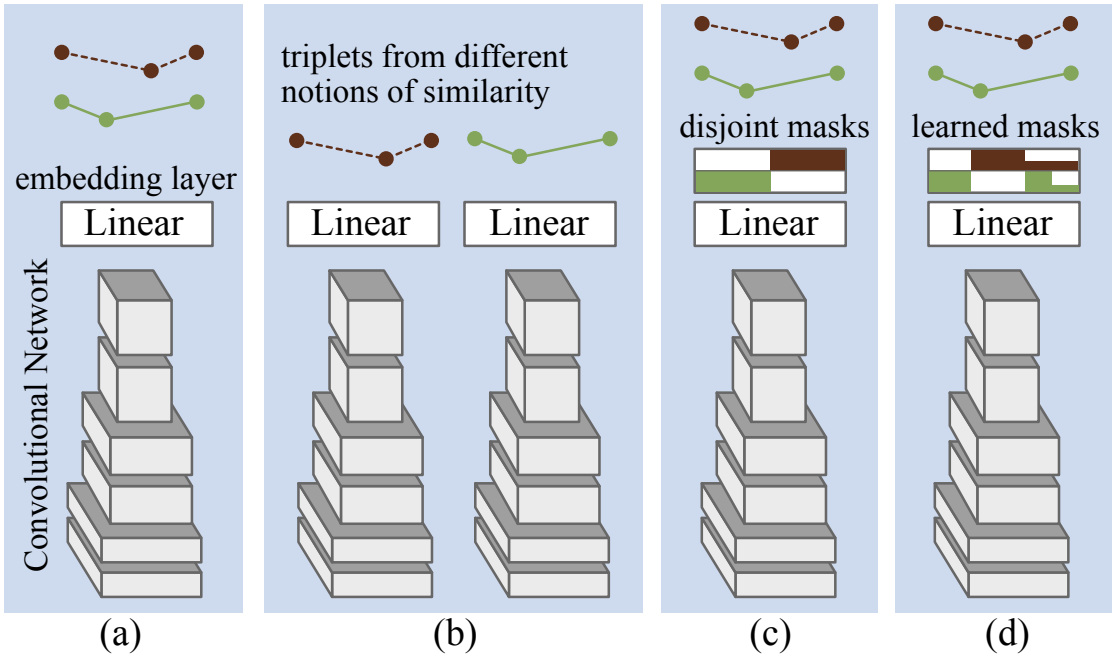


Figure 3.6: We show the four different model variants used in our experiments with the example of three objects being compared according to two contradictory notions of similarity, green and red. (**a**) A standard triplet network that treats all triplets equally (**b**) A set of n_c-many triplet network experts specialized on green or red, respectively (**c**) A CSN with masks pre-set to be disjoint, so that in the embedding each dimension encodes a feature for a specific notion of similarity (**d**) A learned CSN, where the masks are learned to select features relevant to the respective notion of similarity.

different notions of similarity. This ensures the learned embedding is fully disentangled, because each dimension must encode features that describe a specific notion of similarity.

Conditional Similarity Networks - learned masks: The second variant learns the convolutional filters, the embedding and the mask parameters together. This allows the model to learn unique features for the subspaces as well as features shared across tasks. This variant has the additional benefit that the learned masks can provide interesting insight in how different similarity notions are related.

3.3.3 Training Details

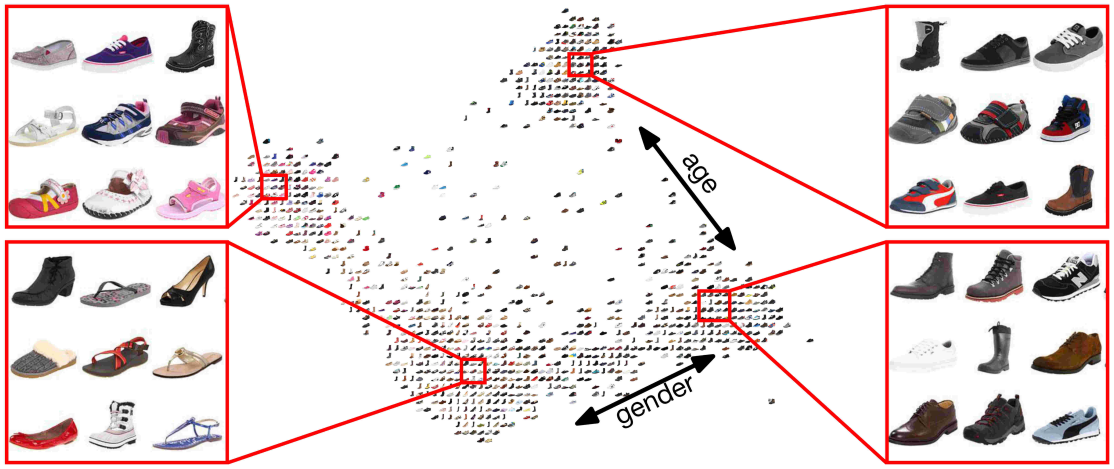
We train different convolutional networks for the two datasets. For the font dataset, we use a variant of the VGG architecture [74] with 9 layers of 3 by 3 convolutions and two fully connected layers, which we train from scratch. For the Zappos dataset we fine-tune an 18 layer deep residual network [32] that is pre-trained on Imagenet [17]. We remove one downsampling module to adjust for the smaller image size. We train the networks with a mini-batch size of 256 and optimize using ADAM [47] with $\alpha = 5\text{E-}5$, $\beta_1 = 0.1$ and $\beta_2 = 0.001$. For all our experiments we use an embedding dimension of 64 and the weights for the embedding losses are $\lambda_1 = 5\text{E-}3$ and $\lambda_2 = 5\text{E-}4$. In each mini-batch we sample triplets uniformly and for each condition in equal proportions. We train each model for 200 epochs and perform early stopping in that we evaluate the snapshot with highest validation performance on the test set.

For our CSN variants, we use two masks over the embedding for the fonts dataset and four masks for the Zappos dataset, one mask per similarity notion. For models with pre-defined masks, we allocate $1/n_c$ th of the embedding dimensions to one task. When learning masks, we initialize β_m using a normal distribution with 0.9 mean and 0.7 variance. Following the ReLU, this results in initial mask values that induce random subspaces for each similarity measure. We observe that different random subspaces perform better than a setup where all subspaces start from the same values. Masks that are initialized as disjoint analogous to the pre-defined masks perform similar to random masks, but are not able to learn shared features.

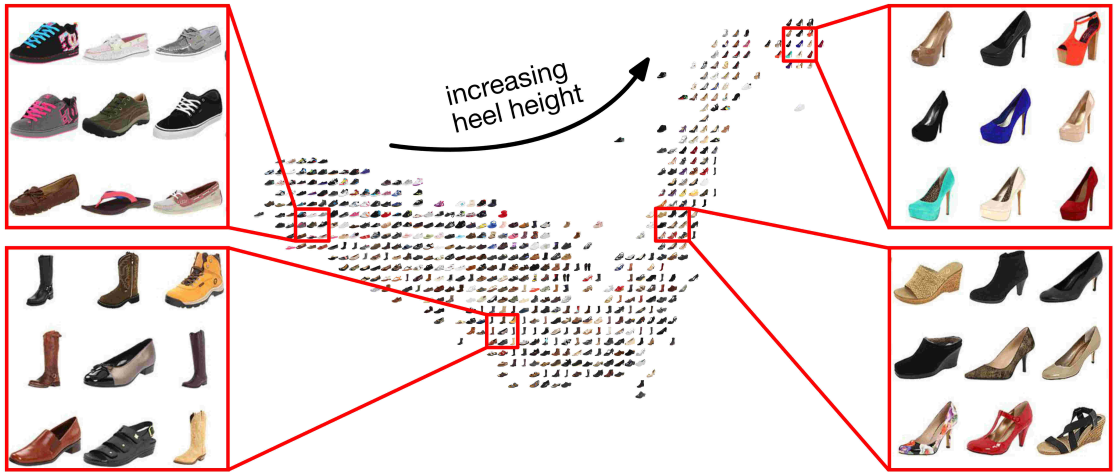
3.3.4 Visual Exploration of the Learned Subspaces

We visually explore the learned embeddings regarding their consistency according to respective similarity notions. We stress that all of these semantic representations are taking place within a shared space produced by the same network. The representations are *disentangled* so that each dimension encodes a feature for a specific notion of similarity. This allows us to use a simple masking operation to look into a specific semantic subspace.

Figure 3.4 shows embeddings of the two subspaces in the Fonts dataset, which we project down to two dimensions using t-SNE [88]. The learned features are successfully disentangled such that the dimensions selected by the first mask describe the character type (left) and those selected by the second mask the font style (right). Figures 3.5 and 3.7 show embeddings of the four subspaces learned with a CSN on the Zappos50k dataset. Figure 3.5(a) shows the subspace encoding features for the *closure mechanism* of the shoes. Figure 3.5(b) shows the subspace attending to the *type* of the shoes. The embedding clearly separates the different types of shoes into boots, slippers and so on. Highlighted areas reveal some interesting details. For example, the highlighted region on the upper right side shows nearby images of the same type ('shoes') that are completely different according to all other aspects. This means the selected feature dimensions successfully focus only on the *type* aspect and do not encode any of the other notions. Figure 3.7(a) shows the subspace for *suggested gender* for the shoes. The subspace separates shoes that are for female and male buyers as well as shoes for adult or youth buyers. The learned submanifold occupies a rotated square with axes defined by gender and age. Finally, Figure 3.7(b) shows a continuous embedding of *heel heights*, which is a subtle visual feature.



(a) Embedding according to the suggested gender



(b) Embedding according to the height of the heels

Figure 3.7: Visualization of the subspaces according to **(a)** suggested gender for the shoes and **(b)** height of the shoes' heel. The result shows that CSNs can learn categorical as well as continuous characteristics at the same time.

3.3.5 Qualitative Analysis Of Subspaces

The key feature of CSNs is the fact that they can learn separated semantic subspaces in the embeddings using the masking mechanism. We visualize the masks for our common model choices in Figure 3.8. We show the traditional triplet loss, where each dimension is equally taken into account for each triplet. Further, we show pre-defined masks that are used to factorize the embedding

into fully disjoint features. Lastly, we show a learned mask. Interestingly, the masks are very sparse in accordance with the 2D embeddings presented in the previous section, confirming that the concepts are low-dimensional. Further, although many additional dimensions are available, the model learned to share some of the features across concepts. This demonstrates that CSNs can learn to only use the required number of dimensions via relevance determination, reducing the need for picking the right embedding dimensionality.

3.3.6 Results on Triplet Prediction

To evaluate the quality of the learned embeddings by the different model variants, we test how well they generalize to unseen triplets. In particular, we perform triplet prediction on a testset of hold-out triplets from the Zappos50k dataset. We first train each model on a fixed set of triplets, where triplets are sourced from the four different notions of similarity. After convergence, we evaluate for each triplet with associated query $\{i, j, l, c\}$ in the testset whether the distance between i and l is smaller than between i and j according to concept/query c. Since this is a binary task, random guessing would perform at an error rate of 50%.

The error rates for the different models are shown in Table 3.1. Standard Triplet Networks fail to capture fine-grained similarity and only reach an error rate of 23.72%. The set of task specific triplet networks greatly improves on that, achieving an error rate of 11.35%. This shows that simply *learning a single space cannot capture multiple similarity notions*. However, this comes at a the cost of n_c times more model parameters. Conditional Similarity Networks with fixed

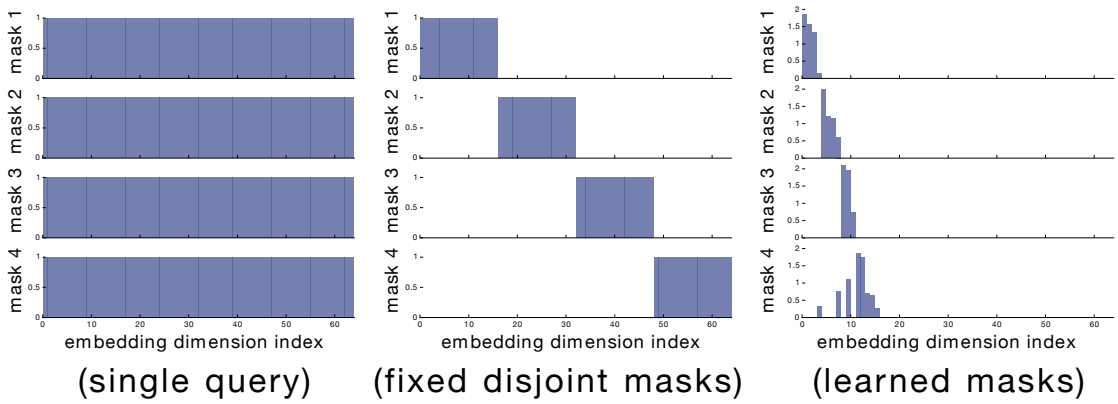


Figure 3.8: Visualization of the masks: **Left:** In standard triplet networks, each dimension is equally taken into account for each triplet. **Center:** The Conditional Similarity Network allows to focus on a subset of the embedding to answer a triplet question. Here, each mask focuses on one fourth. **Right:** For learned masks, it is evident that the model learns to switch off different dimensions per question. Further, a small subset is shared across tasks.

disjoint masks achieve an error rate of 10.79%, clearly outperforming both the single triplet network as well as the set of specialist networks, which have a lot more parameters available for learning. This means by factorizing the embedding space into separate semantic subspaces, CSNs can successfully capture multiple similarity notions without requiring substantially more parameters. Moreover, CSNs benefit from learning all concepts jointly within one model, utilizing shared structure between the concepts while keeping the subspaces separated. CSNs with learned masks achieve an error rate of 10.73% improving performance even further. This indicates the benefits from allowing the model to determine the relevant dimensions and to share features across concepts.

Further, we evaluate the impact of the number of unique triplets available during training on performance. We compare models trained on 5, 12.5 25, 50 and 200 thousand triplets per concept. Figure 3.9 shows that triplet networks generally improve with more available triplets. Further, CSNs with fixed masks

Table 3.1: Triplet Prediction Results: We evaluate how many triplets of the test set are satisfied in the learned embeddings. Triplets come from four different similarity notions. The proposed Conditional Similarity Network clearly outperforms standard triplet networks that treat each triplet as if it came from the same similarity notion. Moreover, CSNs even outperform sets of specialist triplet networks where a lot more parameters are available during training and each network is specifically trained towards one similarity notion. CSNs with learned masks provide the best performance.

Method	Error Rate
Standard Triplet Network	23.72%
Set of Specialized Triplet Networks	11.35%
CSN fixed disjoint masks	10.79%
CSN learned masks	**10.73%**

consistently outperform set of specialized triplet networks. Lastly, CSNs with learned masks generally require more triplets, since they need to learn the embedding as well as the masks. However, when enough triplets are available, they provide the best performance.

3.3.7 Analysis of Convolutional Features Using Off-Task Classification

We now evaluate how the different learning approaches affect the visual features of the networks. We compare standard triplet networks to CSNs. Both are initialized from the same ImageNet pre-trained residual network and fine-tuned using the same triplets and with their respective losses as described in Section 3.3.6. We evaluate the features learned by the two approaches, by subsequently performing brand classification on the Zappos dataset. In particular, we keep all convolutional filters fixed and replace the last embedding layer for both networks with one hidden and one softmax classification layer. We select

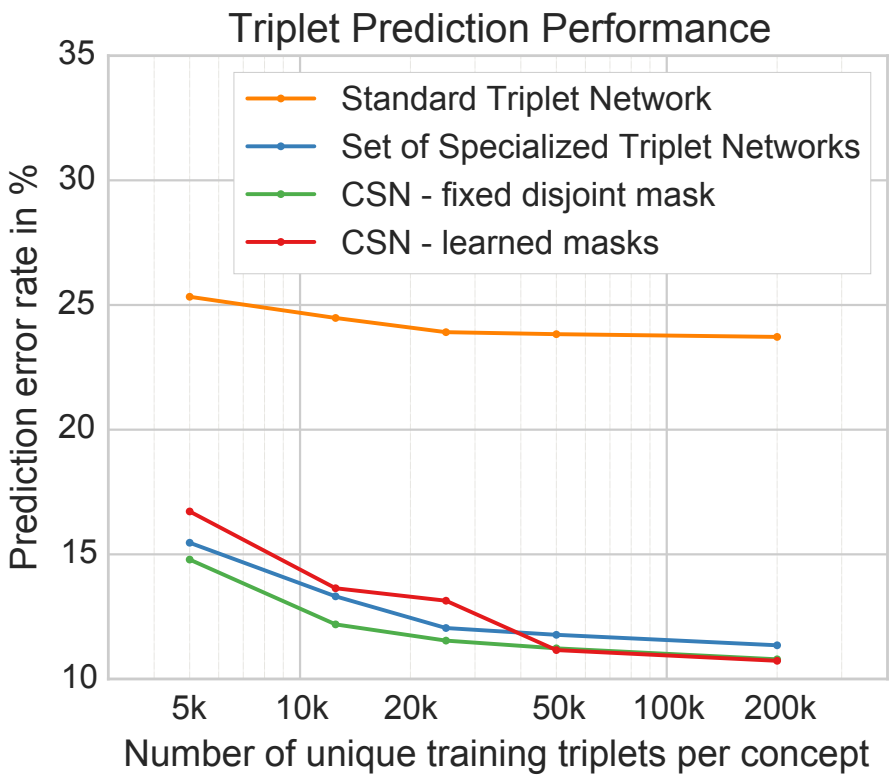


Figure 3.9: Triplet prediction performance with respect to number of unique training triplets available. CSNs with fixed masks consistently outperform the set of specialized triplet networks. CSNs with learned masks generally require more triplets, since they need to learn the embedding as well as the masks. However, when enough triplets are available, they provide the best performance.

the 30 brands in the Zappos dataset with the most examples and train with a standard multi-class classification approach using the 30 brands as classes. It is noteworthy that the triplets used for the fine-tuning do not contain brand information.

The results are shown in Table 3.2. The residual network trained on ImageNet leads to very good initial visual features for general classification tasks. Starting from the pretrained model, we observe that the standard triplet learning approach *decreases* the quality of the visual features, while CSNs retain most

Table 3.2: Using off-task classification, we evaluate how standard triplet networks and CSNs affect the convolutional features of the ImageNet-pretrained network they are based on. Naively training a standard triplet network with triplets from different similarity notions hurts the underlying convolutional features.

Method	Top 1 Accuracy
ResNet trained on ImageNet	54.00%
Standard Triplet Network	49.08%
Conditional Similarity Network	53.67%

of the information. In the triplet prediction experiment in Section 3.3.6 standard triplet networks do not perform well, as they are naturally limited by the fact that contradicting notions cannot be satisfied in one single space. This classification result documents that the problem reaches even deeper. The contradicting gradients do not stop at the embedding layer, instead, they expose the entire network to inconsistent learning signals and hurt the underlying convolutional features.

3.4 Conclusion

In this work, we propose Conditional Similarity Networks to learn nonlinear embeddings which incorporate multiple aspect of similarity within a shared embedding. The learned embeddings are disentangled such that each embedding dimension encodes semantic features for a specific aspect of similarity. This allows to compare objects according to various notions by selecting an appropriate subspace using an element-wise mask. We demonstrate that CSNs clearly outperform single triplet networks, and even sets of specialist triplet networks where a lot more parameters are available and each network is trained

towards one similarity notion.

Further, instead of being a black-box predictor, CSNs are qualitatively highly interpretable as evidenced by our exhibition of the semantic submanifolds they learn. Moreover, they provide a feature-exploration mechanism through the learned masks which surfaces the structure of the private and shared features between the different similarity aspects.

Lastly, we empirically find that naively training a triplet network with triplets generated through different similarity notions does not only limit the ability to correctly embed triplets, it also hurts the underlying convolutional features and thus generalization performance. The proposed CSNs are a simple to implement and easy to train end-to-end alternative to resolve these problems.

For future work, it would be interesting to consider learning from unlabeled triplets with a clustering mechanism to discover similarity substructures in an unsupervised way.

CHAPTER 4

ON THE INDEPENDENCE OF LAYERS IN RESIDUAL NETWORKS

Most modern computer vision systems follow a familiar architecture, processing inputs from low-level features up to task specific high-level features. Recently proposed residual networks [32, 33] challenge this conventional view in three ways. First, they introduce *identity skip-connections* that bypass residual layers, allowing data to flow from any layers directly to any subsequent layers. This is in stark contrast to the traditional strictly sequential pipeline. Second, skip connections give rise to networks that are *two orders of magnitude deeper* than previous models, with as many as 1202 layers. This is contrary to architectures like AlexNet [51] and even biological systems [70] that can capture complex concepts within half a dozen layers.[1] Third, in initial experiments, we observe that *removing single layers from residual networks at test time does not noticeably affect their performance*. This is surprising because removing a layer from a traditional architecture such as VGG [74] leads to a dramatic loss in performance.

In this work we investigate the impact of these differences. To address the influence of identity skip-connections, we introduce the *unraveled view*. This novel representation shows residual networks can be viewed as a collection of many paths instead of a single deep network. Further, the perceived resilience of residual networks raises the question whether the paths are dependent on each other or whether they exhibit a degree of redundancy. To find out, we perform a *lesion study*. The results show ensemble-like behavior in the sense that removing paths from residual networks by deleting layers or corrupting paths by reordering layers only has a modest and smooth impact on performance.

[1] Making the common assumption that a layer in a neural network corresponds to a cortical area.

Finally, we investigate the depth of residual networks. Unlike traditional models, paths through residual networks vary in length. The distribution of path lengths follows a binomial distribution, meaning that the majority of paths in a network with 110 layers are only about 55 layers deep. Moreover, we show most gradient during training comes from paths that are even shorter, i.e., 10-34 layers deep.

This reveals a tension. On the one hand, residual network performance improves with adding more and more layers [33]. However, on the other hand, residual networks can be seen as collections of many paths and the only effective paths are relatively shallow. Our results could provide a first explanation: residual networks do not resolve the vanishing gradient problem by preserving gradient flow throughout the entire depth of the network. Rather, they enable very deep networks by shortening the effective paths. For now, short paths still seem necessary to train very deep networks.

In this paper we make the following contributions:

- We introduce the unraveled view, which illustrates that residual networks can be viewed as a collection of many paths, instead of a single ultra-deep network.
- We perform a lesion study to show that these paths do not strongly depend on each other, even though they are trained jointly. Moreover, they exhibit ensemble-like behavior in the sense that their performance smoothly correlates with the number of valid paths.
- We investigate the gradient flow through residual networks, revealing that only the short paths contribute gradient during training. Deep paths are not required during training.

A preliminary version of this chapter has been published at NIPS 2016 [98].

4.1 Related Work

4.1.1 The Sequential and Hierarchical Computer Vision Pipeline

Visual processing has long been understood to follow a hierarchical process from the analysis of simple to complex features. This formalism is based on the discovery of the receptive field [41], which characterizes the visual system as a hierarchical and feedforward system. Neurons in early visual areas have small receptive fields and are sensitive to basic visual features, e.g., edges and bars. Neurons in deeper layers of the hierarchy capture basic shapes, and even deeper neurons respond to full objects. This organization has been widely adopted in the computer vision and machine learning literature, from early neural networks such as the Neocognitron [22] and the traditional hand-crafted feature pipeline of Malik and Perona [61] to convolutional neural networks [51, 52]. The recent strong results of very deep neural networks [74, 81] led to the general perception that it is the depth of neural networks that govern their expressive power and performance. In this work, we show that residual networks do not necessarily follow this tradition.

4.1.2 Residual Networks

Residual networks [32, 33] are neural networks in which each layer consists of a residual module f_i and a skip connection[2] bypassing f_i. Since layers in residual networks can comprise multiple convolutional layers, we refer to them as residual blocks in the remainder of this paper. For clarity of notation, we omit the initial pre-processing and final classification steps. With y_{i-1} as is input, the output of the ith block is recursively defined as

$$y_i \equiv f_i(y_{i-1}) + y_{i-1}, \tag{4.1}$$

where $f_i(x)$ is some sequence of convolutions, batch normalization [42], and Rectified Linear Units (ReLU) as nonlinearities. Figure 4.1 (a) shows a schematic view of this architecture. In the most recent formulation of residual networks [33], $f_i(x)$ is defined by

$$f_i(x) \equiv W_i \cdot \sigma\left(B\left(W_i' \cdot \sigma\left(B\left(x\right)\right)\right)\right), \tag{4.2}$$

where W_i and W_i' are weight matrices, $\cdot$ denotes convolution, $B(x)$ is batch normalization and $\sigma(x) \equiv \max(x, 0)$. Other formulations are typically composed of the same operations, but may differ in their order.

The idea of branching paths in neural networks is not new. For example, in the regime of convolutional neural networks, models based on inception modules [81] were among the first to arrange layers in blocks with parallel paths rather than a strict sequential order. We choose residual networks for this study because of their simple design principle.

[2] We only consider identity skip connections, but this framework readily generalizes to more complex projection skip connections when downsampling is required.

4.1.3 Highway Networks

Residual networks can be viewed as a special case of highway networks [77]. The output of each layer of a highway network is defined as

$$y_{i+1} \equiv f_{i+1}(y_i) \cdot t_{i+1}(y_i) + y_i \cdot (1 - t_{i+1}(y_i)) \tag{4.3}$$

This follows the same structure as Equation (4.1). Highway networks also contain residual modules and skip connections that bypass them. However, the output of each path is attenuated by a gating function t, which has learned parameters and is dependent on its input. Highway networks are equivalent to residual networks when $t_i(\cdot) = 0.5$, in which case data flows equally through both paths. Given an omnipotent solver, highway networks could learn whether each residual module should affect the data. This introduces more parameters and more complexity.

4.1.4 Investigating Neural Networks

Several investigative studies seek to better understand convolutional neural networks. For example, Zeiler and Fergus [108] visualize convolutional filters to unveil the concepts learned by individual neurons. Further, Szegedy et al. [82] investigate the function learned by neural networks and how small changes in the input called adversarial examples can lead to large changes in the output. Within this stream of research, the closest study to our work is from Yosinski et al. [106], which performs lesion studies on AlexNet. They discover that early layers exhibit little co-adaptation and later layers have more co-adaptation. These papers, along with ours, have the common thread of ex-

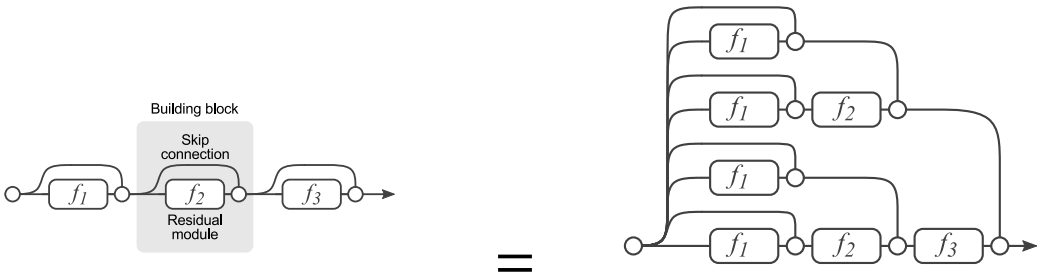
ploring specific aspects of neural network performance. In our study, we focus our investigation on structural properties of neural networks.

4.1.5 Ensembling

Since the early days of neural networks, researchers have used simple ensembling techniques to improve performance. Though boosting has been used in the past [68], one simple approach is to arrange a committee [19] of neural networks in a simple voting scheme, where the final output predictions are averaged. Top performers in several competitions use this technique almost as an afterthought [33, 51, 74]. Generally, one key characteristic of ensembles is their smooth performance with respect to the number of members. In particular, the performance increase from additional ensemble members gets smaller with increasing ensemble size. Even though they are not strict ensembles, we show that residual networks behave similarly.

4.1.6 Dropout

Hinton et al. [76] show that dropping out individual neurons during training leads to a network that is equivalent to averaging over an ensemble of exponentially many networks. Similar in spirit, stochastic depth [39] trains an ensemble of networks by dropping out entire layers during training. In this work, we show that one does not need a special training strategy such as stochastic depth to drop out layers. Entire layers can be removed from plain residual networks without impacting performance, indicating that they do not strongly depend on



(a) Conventional 3-block residual network

(b) Unraveled view of (a)

Figure 4.1: Residual Networks are conventionally shown as (a), which is a natural representation of Equation (4.1). When we expand this formulation to Equation (4.6), we obtain an *unraveled view* of a 3-block residual network (b). Circular nodes represent additions. From this view, it is apparent that residual networks have $O(2^n)$ implicit paths connecting input and output and that adding a block doubles the number of paths.

each other.

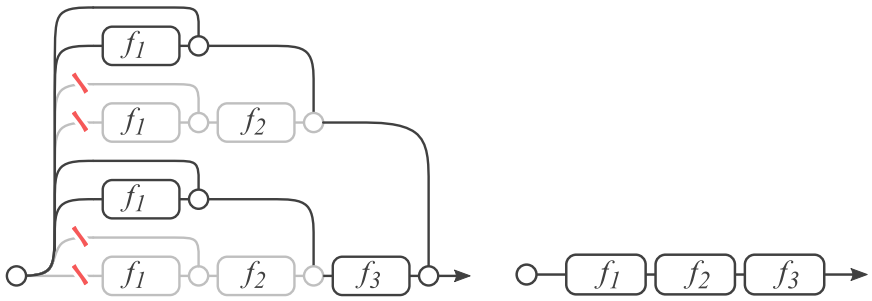
4.2 The Unraveled View of Residual Networks

To better understand residual networks, we introduce a formulation that makes it easier to reason about their recursive nature. Consider a residual network with three building blocks from input y_0 to output y_3. Equation (4.1) gives a recursive definition of residual networks. The output of each stage is based on the combination of two subterms. We can make the shared structure of the residual network apparent by unrolling the recursion into an exponential number of nested terms, expanding one layer at each substitution step:

$$y_3 = y_2 + f_3(y_2) \tag{4.4}$$

$$= [y_1 + f_2(y_1)] \quad + \quad f_3(y_1 + f_2(y_1)) \tag{4.5}$$

$$= [y_0 + f_1(y_0) + f_2(y_0 + f_1(y_0))] \; + \; f_3(y_0 + f_1(y_0) + f_2(y_0 + f_1(y_0))) \tag{4.6}$$



(a) Deleting f_2 from unraveled view

(b) Ordinary feedforward network

Figure 4.2: Deleting a layer in residual networks at test time (a) is equivalent to zeroing half of the paths. In ordinary feed-forward networks (b) such as VGG or AlexNet, deleting individual layers alters the only viable path from input to output.

We illustrate this expression tree graphically in Figure 4.1 (b). With subscripts in the function modules indicating weight sharing, this graph is equivalent to the original formulation of residual networks. The graph makes clear that data flows along many *paths* from input to output. Each path is a unique configuration of which residual module to enter and which to skip. Conceivably, each unique path through the network can be indexed by a binary code $b \in \{0,1\}^n$ where $b_i = 1$ iff the input flows through residual module f_i and 0 if f_i is skipped. It follows that residual networks have 2^n paths connecting input to output layers.

In the classical visual hierarchy, each layer of processing depends *only* on the output of the previous layer. Residual networks cannot strictly follow this pattern because of their inherent structure. Each module $f_i(\cdot)$ in the residual network is fed data from a mixture of 2^{i-1} different distributions generated from every possible configuration of the previous $i-1$ residual modules.

Compare this to a strictly sequential network such as VGG or AlexNet, de-

picted conceptually in Figure 4.2 (b). In these networks, input always flows from the first layer straight through to the last in a single path. Written out, the output of a three-layer feed-forward network is

$$y_3^{FF} = f_3^{FF}(f_2^{FF}(f_1^{FF}(y_0))) \tag{4.7}$$

where $f_i^{FF}(x)$ is typically a convolution followed by batch normalization and ReLU. In these networks, each f_i^{FF} is only fed data from a single path configuration, the output of $f_{i-1}^{FF}(\cdot)$.

It is worthwhile to note that ordinary feed-forward neural networks can also be "unraveled" using the above thought process at the level of individual neurons rather than layers. This renders the network as a collection of different paths, where each path is a unique configuration of neurons from each layer connecting input to output. Thus, all paths through ordinary neural networks are of the same length. However, paths in residual networks have varying length. Further, each path in a residual network goes through a different subset of layers.

Based on these observations, we formulate the following questions and address them in our experiments below. Are the paths in residual networks dependent on each other or do they exhibit a degree of redundancy? If the paths do not strongly depend on each other, do they behave like an ensemble? Do paths of varying lengths impact the network differently?

4.3 Lesion Study

In this section, we use three lesion studies to show that paths in residual networks do not strongly depend on each other and that they behave like an ensemble. All experiments are performed at test time on CIFAR-10 [50]. Experiments on ImageNet [17] show comparable results. We train residual networks with the standard training strategy, dataset augmentation, and learning rate policy, [33]. For our CIFAR-10 experiments, we train a 110-layer (54-module) residual network with modules of the "pre-activation" type which contain batch normalization as first step. For ImageNet we use 200 layers (66 modules). It is important to note that we did not use any special training strategy to adapt the network. In particular, we did *not* use any perturbations such as stochastic depth during training.

4.3.1 Deleting Individual Layers in Neural Networks at Test-Time

As a motivating experiment, we will show that not all transformations within a residual network are necessary by deleting individual modules from the neural network after it has been fully trained. To do so, we remove the residual module from a single building block, leaving the skip connection (or downsampling projection, if any) untouched. That is, we change $y_i = y_{i-1} + f_i(y_{i-1})$ to $y'_i = y_{i-1}$. We can measure the importance of each building block by varying which residual module we remove. To compare to conventional convolutional neural networks, we train a VGG network with 15 layers, setting the number of

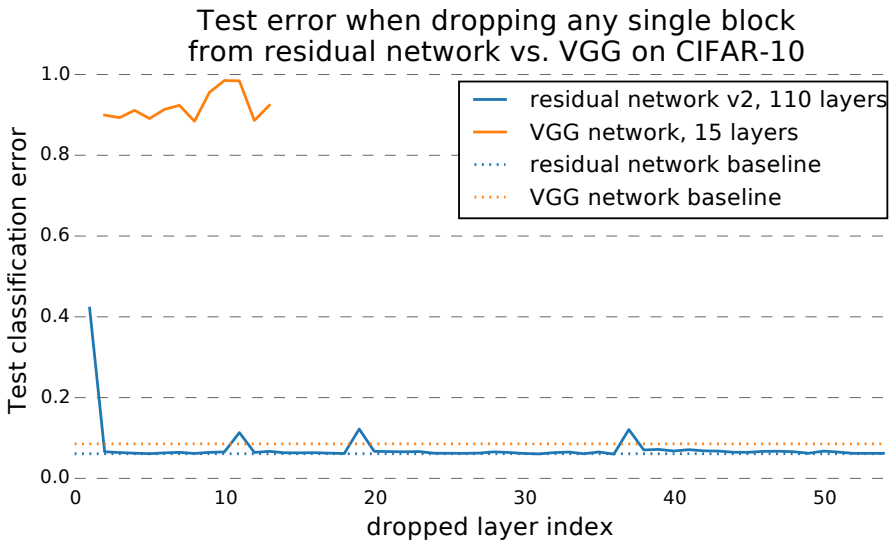


Figure 4.3: Deleting individual layers from VGG and a residual network on CIFAR-10. VGG performance drops to random chance when any one of its layers is deleted, but deleting individual modules from residual networks has a minimal impact on performance. Removing downsampling modules has a slightly higher impact.

channels to 128 for all layers to allow the removal of any layer.

It is unclear whether any neural network can withstand such a drastic change to the model structure. We expect them to break because dropping any layer drastically changes the input distribution of all subsequent layers.

The results are shown in Figure 4.3. As expected, deleting any layer in VGG reduces performance to chance levels. Surprisingly, **this is not the case for residual networks**. Removing downsampling blocks does have a modest impact on performance (peaks in Figure 4.3 correspond to downsampling building blocks), but no other block removal lead to a noticeable change. This result shows that to some extent, the structure of a residual network can be changed at runtime without affecting performance. Experiments on ImageNet show comparable results, as seen in Figure 4.4.

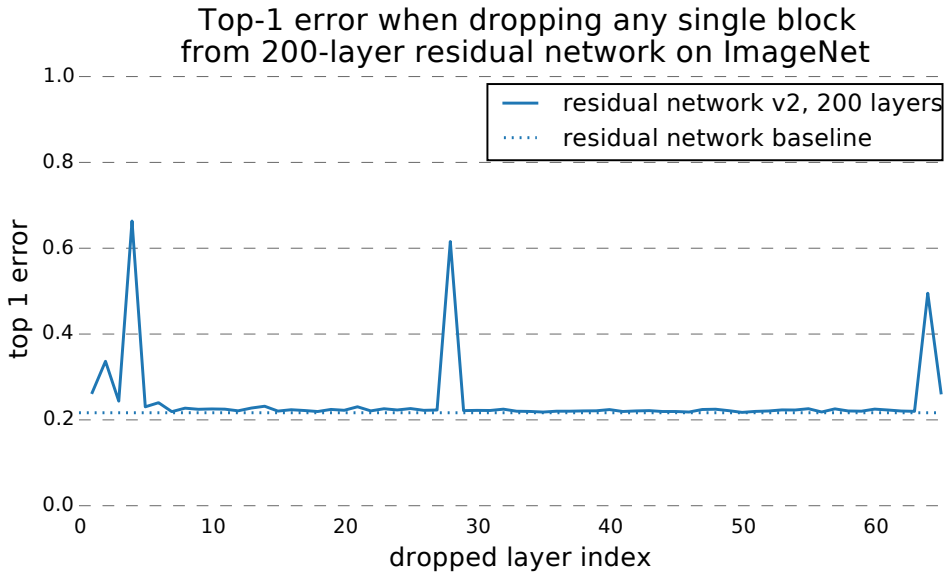
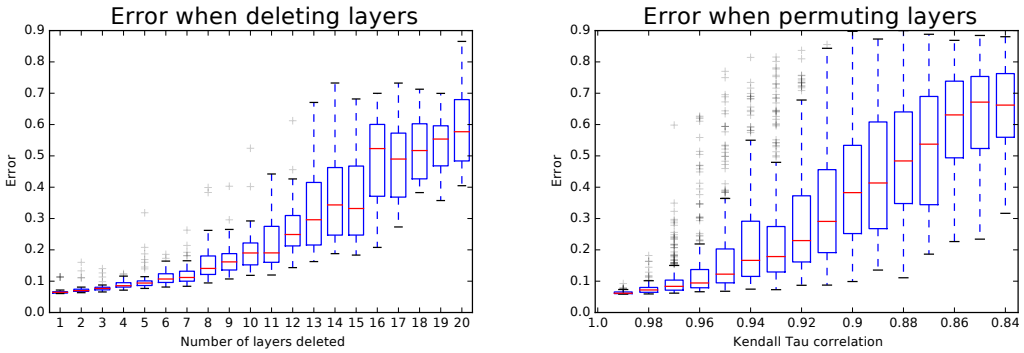


Figure 4.4: Results when dropping individual blocks from residual networks trained on ImageNet are similar to CIFAR results. However, downsampling layers tend to have more impact on ImageNet.

Why are residual networks resilient to dropping layers but VGG is not? Expressing residual networks in the unraveled view provides a first insight. It shows that residual networks can be seen as a collection of many paths. As illustrated in Figure 4.2 (a), when a layer is removed, the number of paths is reduced from 2^n to 2^{n-1}, leaving half the number of paths valid. VGG only contains a single usable path from input to output. Thus, when a single layer is removed, the only viable path is corrupted. *This result suggests that paths in a residual network do not strongly depend on each other although they are trained jointly.*

4.3.2 Deleting Many Modules in Residual Networks at Test-Time

Having shown that paths do not strongly depend on each other, we investigate whether the collection of paths shows ensemble-like behavior. One key charac-



(a) (b)

Figure 4.5: (a) Error increases smoothly when randomly deleting several modules from a residual network. (b) Error also increases smoothly when reordering a residual network by shuffling building blocks. The degree of reordering is measured by the Kendall Tau correlation coefficient. These results are similar to what one would expect from ensembles.

teristic of ensembles is that their performance depends smoothly on the number of members. If the collection of paths were to behave like an ensemble, we would expect test-time performance of residual networks to smoothly correlate with the *number of valid paths*. This is indeed what we observe: deleting increasing numbers of residual modules, increases error smoothly, Figure 4.5 (a). This implies residual networks behave like ensembles.

When deleting k residual modules from a network originally of length n, the number of valid paths decreases to $O(2^{n-k})$. For example, the original network started with 54 building blocks, so deleting 10 blocks leaves 2^{44} paths. Though the collection is now a factor of roughly 10^{-6} of its original size, there are still many valid paths and error remains around 0.2.

4.3.3 Reordering Modules in Residual Networks at Test-Time

Our previous experiments were only about dropping layers, which have the effect of removing paths from the network. In this experiment, we consider changing the structure of the network by *re-ordering* the building blocks. This has the effect of removing some paths and inserting new paths that have never been seen by the network during training. In particular, it moves high-level transformations before low-level transformations.

To re-order the network, we swap k randomly sampled pairs of building blocks with compatible dimensionality, ignoring modules that perform down-sampling. We graph error with respect to the Kendall Tau rank correlation coefficient which measures the amount of corruption. The results are shown in Figure 4.5 (b). As corruption increases, the error smoothly increases as well. This result is surprising because it suggests that residual networks can be re-configured to some extent at runtime.

4.4 The Importance of Short Paths in Residual Networks

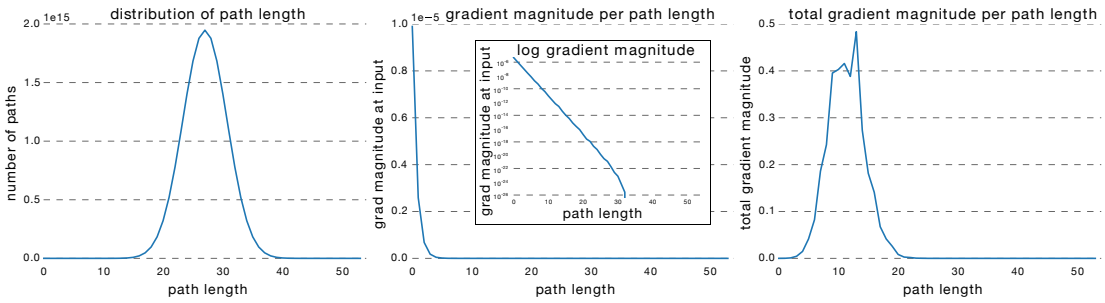
Now that we have seen that there are many paths through residual networks and that they do not necessarily depend on each other, we investigate their characteristics.

4.4.1 Distribution of Path Lengths

Not all paths through residual networks are of the same length. For example, there is precisely one path that goes through all modules and n paths that go only through a single module. From this reasoning, the distribution of all possible path lengths through a residual network follows a Binomial distribution. Thus, we know that the path lengths are closely centered around the mean of $n/2$. Figure 4.6 (a) shows the path length distribution for a residual network with 54 modules; more than 95% of paths go through 19 to 35 modules.

4.4.2 Vanishing Gradients in Residual Networks

Generally, data flows along all paths in residual networks. However, not all paths carry the same amount of gradient. In particular, the length of the paths through the network affects the gradient magnitude during backpropagation [9, 34]. To empirically investigate the effect of vanishing gradients on residual networks we perform the following experiment. Starting from a trained network with 54 blocks, we sample *individual paths* of a certain length and measure the norm of the gradient that arrives at the input. To sample a path of length k, we first feed a batch forward through the whole network. During the backward pass, we randomly sample k residual blocks. For those k blocks, we only propagate through the residual module; for the remaining $n - k$ blocks, we only propagate through the skip connection. Thus, we only measure gradients that flow through the single path of length k. We sample 1,000 measurements for each length k using random batches from the training set. The results show that the gradient magnitude of a path decreases exponentially with the number



(a) (b) (c)

Figure 4.6: How much gradient do the paths of different lengths contribute in a residual network? To find out, we first show the distribution of all possible path lengths (a). This follows a Binomial distribution. Second, we record how much gradient is induced on the first layer of the network through paths of varying length (b), which appears to decay roughly exponentially with the number of modules the gradient passes through. Finally, we can multiply these two functions (c) to show how much gradient comes from all paths of a certain length. Though there are many paths of medium length, paths longer than ~20 modules are generally too long to contribute noticeable gradient during training. This suggests that the effective paths in residual networks are relatively shallow.

of modules it went through in the backward pass, Figure 4.6 (b).

4.4.3 The Effective Paths in Residual Networks Are Relatively Shallow

Finally, we can use these results to deduce whether shorter or longer paths contribute most of the gradient during training. To find the total gradient magnitude contributed by paths of each length, we multiply the frequency of each path length with the expected gradient magnitude. The result is shown in Figure 4.6 (c). Surprisingly, almost all of the gradient updates during training come from paths between 5 and 17 modules long, even though they constitute only 0.45% of all paths through this network. Moreover, in comparison to the total

length of the network, these paths are *relatively shallow*. We refer to these paths as *effective paths*.

To validate this result, we retrain a residual network from scratch that only sees the effective paths during training. This ensures that no long path is ever used. If the retrained model is able to perform competitively compared to training the full network, we know that long paths in residual networks are not needed during training. We achieve this by only training a subset of the modules during each mini batch. In particular, we choose the number of modules such that the distribution of paths during training aligns with the distribution of the effective paths in the whole network. For the network with 54 modules, this means we sample exactly 23 modules during each training batch. Then, the path lengths during training are centered around 11.5 modules, well aligned with the effective paths. In our experiment, the network trained only with the effective paths achieves a 5.96% error rate, whereas the full model achieves a 6.10% error rate. There is no statistically significant difference. This demonstrates that indeed only the effective paths are needed.

4.5 Discussion

Removing residual modules mostly removes long paths Deleting a module from a residual network mainly removes the long paths through the network. In particular, when deleting d residual modules from a network of length n, the fraction of paths remaining per path length x is given by

$$\text{fraction of remaining paths of length } x = \frac{\binom{n-d}{x}}{\binom{n}{x}} \tag{4.8}$$

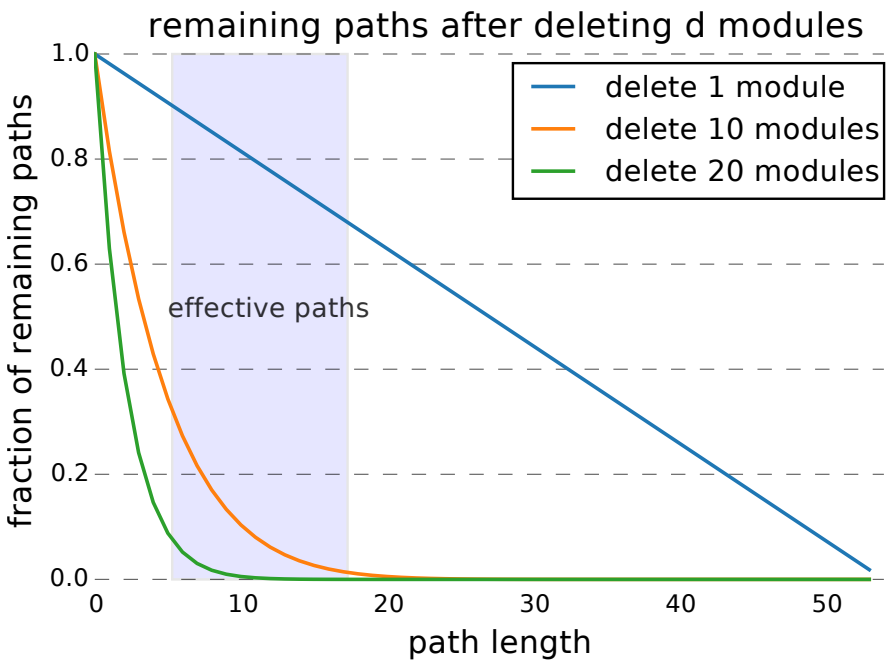


Figure 4.7: Fraction of paths remaining after deleting individual layers. Deleting layers mostly affects long paths through the networks.

Figure 4.7 illustrates the fraction of remaining paths after deleting 1, 10 and 20 modules from a 54 module network. It becomes apparent that the deletion of residual modules mostly affects the long paths. Even after deleting 10 residual modules, many of the *effective paths* between 5 and 17 modules long are still valid. Since mainly the effective paths are important for performance, this result is in line with the experiment shown in Figure 4.5 (a). Performance only drops slightly up to the removal of 10 residual modules, however, for the removal of 20 modules, we observe a severe drop in performance.

Connection to highway networks In highway networks, $t_i(\cdot)$ multiplexes data flow through the residual and skip connections and $t_i(\cdot) = 0.5$ means both paths are used equally. For highway networks in the wild, [77] observe empirically that the gates commonly deviate from $t_i(\cdot) = 0.5$. In particular, they tend to be biased toward sending data through the skip connection; in other words,

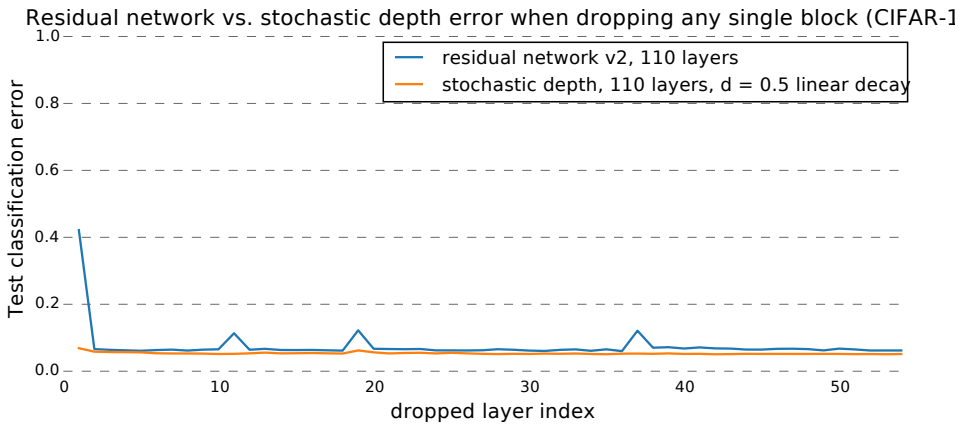


Figure 4.8: Impact of stochastic depth on resilience to layer deletion. Training with stochastic depth only improves resilience slightly, indicating that plain residual networks already don't depend on individual layers. Compare to Fig. 4.3.

the network learns to use short paths. Similar to our results, it reinforces the importance of short paths.

Effect of stochastic depth training procedure Recently, an alternative training procedure for residual networks has been proposed, referred to as stochastic depth [39]. In that approach a random subset of the residual modules is selected for each mini-batch during training. The forward and backward pass is only performed on those modules. Stochastic depth does not affect the number of paths in the network because all paths are available at test time. However, it changes the distribution of paths seen during training. In particular, mainly short paths are seen. Further, by selecting a different subset of short paths in each mini-batch, it encourages the paths to produce good results independently.

Does this training procedure significantly reduce the dependence between paths? We repeat the experiment of deleting individual modules for a residual network trained using stochastic depth. The result is shown in Figure 4.8. Training with stochastic depth improves resilience slightly; only the dependence on

the downsampling layers seems to be reduced. By now, this is not surprising: we know that plain residual networks already don't depend on individual layers.

4.6 Conclusion

What is the reason behind residual networks increased performance? In the most recent iteration of residual networks, He et al. [33] provide one hypothesis: We obtain these results via a simple but essential concept—going deeper. While it is true that they are deeper than previous approaches, we present a complementary explanation. First, our unraveled view reveals that residual networks can be viewed as a collection of many paths, instead of a single ultra deep network. Second, we perform lesion studies to show that, although these paths are trained jointly, they do not strongly depend on each other. Moreover, they exhibit ensemble-like behavior in the sense that their performance smoothly correlates with the number of valid paths. Finally, we show that the paths through the network that contribute gradient during training are shorter than expected. In fact, deep paths are not required during training as they do not contribute any gradient. Thus, residual networks do not resolve the vanishing gradient problem by preserving gradient flow throughout the entire depth of the network. This insight reveals that depth is still an open research question. These promising observations provide a new lens through which to examine neural networks.

CHAPTER 5

CONVOLUTIONAL NETWORKS WITH INPUT-CONDITIONAL TOPOLOGY

Often, convolutional networks (Convnets) are already confident about the high-level concept of an image after computing only a few layers. This raises the question of what happens in the remainder of the network that often comprises hundreds of layers for many state-of-the-art models. To shed light on this, it is important to note that due to their success, Convnets are used to classify increasingly large sets of visually diverse categories. As a consequence, most parameters are used to model high-level features, because in contrast to low-level and many mid-level concepts, high-level features are not broadly shared across categories. As a result, the networks become larger and slower as the number of categories rises. Moreover, for any given input image the number of computed features focusing on unrelated concepts increases.

What if, after identifying that an image contains a bird, a Convnet could move directly to a layer that can distinguish different bird species, without executing intermediate layers that specialize in unrelated aspects? Intuitively, the more the network already knows about an image, the better it could be at deciding which layer to compute next. Such a network would be able to decouple inference time from the number of learned concepts. Our study in Chapter 4 provides a key insight towards the realization of this scenario. There we show that for residual networks (Resnets) [32], although all layers are trained jointly, they exhibit a high degree of independence. In fact, almost any individual layer can be removed from a trained Resnet without interfering with other layers. Resnets are Convnets with additional identity skip-connections that bypass each layer.

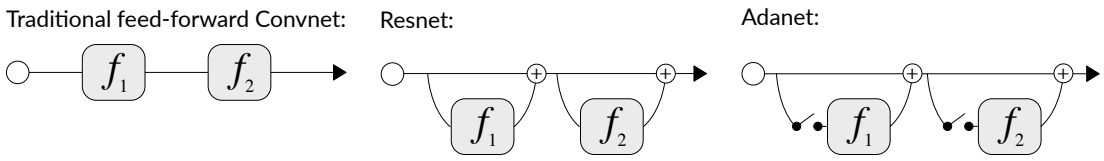


Figure 5.1: Adanets (right) follow a high level structure similar to Resnets (center). The key difference is that for each layer, a gate determines whether to execute or skip the layer. This enables individual computational graphs conditioned on the input.

This leads us to the following research question: *Do we really need fixed structures for convolutional networks, or could we assemble network graphs on the fly, conditioned on the input?*

In this work, we propose Adanets, convolutional networks that adaptively define their network topology conditioned on the input image. Specifically, Adanets learn a set of convolutional layers and decide for each input image which layers are needed. By learning both general layers useful to all images and expert layers specializing on subsets of categories, Adanets allow only computing features relevant to the input image. It is worthy to note that Adanets do not require special supervision about label hierarchies and relationships.

Figure 5.1 gives a schematic overview of the proposed approach. From a high-level, Adanets follow a similar structure as Resnets. The key difference is that for each residual layer, a gate determines whether the layer is needed for the current input image. The main challenge is that the gates need to make discrete decisions, which are difficult to integrate into convolutional networks that we would like to train using gradient descent. To incorporate the discrete decisions, we build upon recent work [8, 43, 60] that introduces differentiable approximations for discrete stochastic nodes in neural networks. In particular, we model the gates as discrete random variables over two states: to execute the

respective layer or to skip it. Further, we model them conditional on the output of the previous layer. This allows the construction of computation graphs adaptively based on the input image and to train both the convolutional weights as well as the discrete gates jointly in an end-to-end fashion.

In experiments on ImageNet [17], we demonstrate that Adanets effectively learn to generate computational graphs such that for each input only relevant features are computed. In terms of classification accuracy both Adanet 50 and Adanet 101 outperform their Resnet counterpart, while at the same time using 20% and 33% less computations respectively. We further show that, without specific supervision, Adanets discover parts of the class hierarchy and learn specialized layers focusing on subsets of categories such as animals and man-made objects. Adanets even learn distinct inference graphs for some mid-level categories such as birds, dogs and reptiles. Our results demonstrate that by grouping parameters for related classes and only executing relevant layers, Adanets both improve efficiency and also overall classification quality. Lastly, we also study the effect of the adaptive network topologies on the susceptibility towards adversarial examples. We show that Adanets are consistently more robust than Resnets, independent of adversary strength and that the additional robustness persists even when applying additional defense mechanisms.

A preliminary version of this chapter has been made available to the public [94].

5.1 Related Work

5.1.1 Neural Network Composition

Our study is related to prior work in multiple fields. Several works have focused on neural network composition for visual question answering (VQA) [3, 4, 44] and zero-shot learning [63]. While these approaches include convolutional networks, they focus on constructing a fixed computational graph up front to solve tasks such as VQA. In contrast, the focus of our work is to construct a convolutional network conditioned on the input image on the fly during execution.

5.1.2 Adaptive Computation

Our approach can be seen as an example of adaptive computation for neural networks. Cascaded classifiers [99] have a long tradition for computer vision by quickly rejecting "easy" negatives. Recently, similar approaches have been proposed for neural networks [55, 105]. In an alternative direction, [7, 72] propose to adjust the amount of computation in fully-connected neural networks. To adapt the computation time in convolutional networks, [37, 84] propose network architectures that add classification branches to intermediate layers. This allows stopping a computation early once a satisfying level of confidence is reached. Most closely related to our approach is the work on spatially adaptive computation time for residual networks [21]. In that paper, a Resnet adaptively determines after which layer to stop computation. Our work differs from this approach in that we do not perform early stopping, but instead determine

which subset of layers to execute. This is key as it allows the grouping of parameters that are relevant for similar categories and thus enables distinct computational graphs for different categories.

5.1.3 Regularization with Stochastic Noise

Our work is further related to network regularization with stochastic noise. By randomly dropping individual neurons during training, Dropout [76] offers an effective way to prevent neural networks from over-fitting. Closely related is the work on stochastic depth [39], where entire layers of a Resnet are randomly removed during each training iteration. Our work resembles this approach in the sense that it also includes stochastic nodes that decide whether to execute layers. In contrast to our work, layer removal in stochastic depth is independent from the input image and aims to *increase* redundancy among layers. In our work, we construct the computational graph conditioned on the input image to *reduce* redundancy and allow the network to learn layers specialized on subsets of the data.

5.1.4 Attention Mechanisms

Lastly, our work can also be seen as an example of an attention mechanism in that we select specific layers of importance for each input image to assemble the computation graph. This is related to approaches such as highway networks [77] and squeeze-and-excitation networks [36] where the output of a residual layer is rescaled according to the layer's importance. This allows

these approaches to emphasize some layers and pay less attention to others. In contrast to our work, these are soft attention mechanisms and still require the execution of every single layer. Our work is a hard attention mechanism and thus enables decoupling computation time from the number of categories.

5.2 Adanets

Traditional feed-forward Convnets can be considered as a set of N layers which are sequentially applied to an input image. Formally, let $\mathcal{F}_l(\cdot)$, $l \in \{1, ..., N\}$ denote the function computed by the l^{th} layer. With $\mathbf{x}_0$ as input image and $\mathbf{x}_l$ as output of the l^{th} layer, such a network can be recursively defined as

$$\mathbf{x}_l = \mathcal{F}_l(\mathbf{x}_{l-1}) \tag{5.1}$$

Resnets [32] change this definition by introducing identity skip-connections that bypass each layer, i.e., the input to each layer is also added to its output. This has been shown to greatly ease optimization during training. As gradients can propagate directly through the skip-connection, early layers still receive sufficient learning signal even in very deep networks. A Resnet can be defined as

$$\mathbf{x}_l = \mathbf{x}_{l-1} + \mathcal{F}_l(\mathbf{x}_{l-1}) \tag{5.2}$$

In Chapter 4 we studied the effects of the skip-connection and have shown that, although all layers are trained jointly, they exhibit a high degree of independence. Further, almost any individual layer can be removed from a trained Resnet without harming performance.

5.2.1 Adaptive Computation Graph

Inspired by the observation that individual layers can be removed without interfering with other layers, we design Adanet, a network that can define its topology on the fly. Adanet follows the basic structure of a Resnet with the key difference that instead of executing all layers, it determines which subset of layers to execute. In particular, with layers focusing on different subgroups of categories, Adanet can select only those layers necessary for the specific input. An Adanet can be defined as

$$\begin{aligned} \mathbf{x}_l &= \mathbf{x}_{l-1} + z(\mathbf{x}_{l-1}) \cdot \mathcal{F}_l(\mathbf{x}_{l-1}) \\ &\text{where } z(\mathbf{x}_{l-1}) \in \{0, 1\} \end{aligned} \tag{5.3}$$

where $z(\mathbf{x}_{l-1})$ is a gate that, conditioned on the input to the layer, decides whether to execute the next layer. The gate chooses between two discrete states: 0 for 'off' and 1 for 'on', which can be seen as a *hard attention mechanism*.

This is in contrast to related work on soft attention mechanisms such as highway networks [77], which share surface resemblance with our formulation. For highway networks, the output of a layer can be rescaled to emphasize certain layers and reduce attention to others. In contrast to our work, soft attention mechanisms still require the execution of every single layer.

5.2.2 Gating Unit

For the gate to be effective, it needs to (1) understand the input features, (2) make a discrete decision, and (3) operate with low computational overhead. We propose a gate that consists of two main components. The first component estimates the probability for the respective layer to execute. The second component

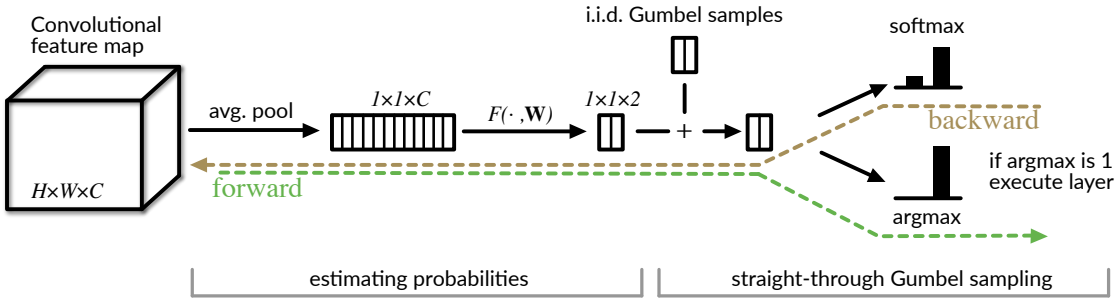


Figure 5.2: **Overview of gating unit.** Each gate comprises two parts. The first part estimates the probability for the layer to be executed. The second part decides whether to execute the layer by sampling from the estimated probability. In particular, the Gumbel-Max trick is used to sample from the discrete distribution. To allow for the propagation of gradients through the discrete decision, we use the Gumbel-Softmax relaxation in the backward pass.

takes the estimated probability and draws a discrete sample from it. Figure 5.2 provides on overview of the gating unit.

Estimating Probabilities

The input to the gate is the output of the previous layer $\mathbf{x}_{l-1} \in \mathbb{R}^{W \times H \times C}$. Since operating on the full feature map is computationally expensive, we build upon recent studies [36, 40, 57] which show that much of the information in convolutional features is captured by the statistics of the different channels and their interdependencies. In particular, we only consider channel-wise means gathered by global average pooling. This compresses the input features into a $1 \times 1 \times C$ channel descriptor.

$$z_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} x_{i,j,c} \tag{5.4}$$

To capture the dependencies between channels, we add a simple non-linear function of two fully-connected layers connected with a ReLU [23] activation function. The output of that operation is a vector β containing the log-

probabilities for computing and skipping the following layer, respectively.

$$\beta = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{z}) \tag{5.5}$$

where σ refers to the ReLU, $\mathbf{W}_1 \in \mathbb{R}^{d\times C}$, $\mathbf{W}_2 \in \mathbb{R}^{2\times d}$ and d is the dimension of the hidden layer. The lightweight design of the gating function leads to minimal computational overhead. For an Adanet based on Resnet 101 for ImageNet, the gating function adds only a computational overhead of 0.04%, but allows to skip 33% of layers on average.

Greedy Gumbel Sampling

After estimating the log-probabilities β, the gating function $z(\mathbf{x}_{l-1})$ needs to make a discrete decision. To incorporate the discrete decision into a convolutional architecture that we would like to train using gradient descent, we build upon recent work that propose approaches for propagating gradients trough stochastic neurons [8, 48]. In particular, we utilize the Gumbel-Max trick [28] and its recent continuous relaxation [43, 60].

A random variable G follows a Gumbel distribution if $G = -\log(-\log(U))$, with U being a sample from the uniform distribution $U \sim \text{Unif}[0, 1]$. With the Gumbel-Max trick we can parameterize discrete distributions in terms of Gumbel random variables. In particular, let X be a discrete random variable with probabilities $P(X = k) \propto \alpha_k$ and let $\{G_k\}_{k\in\{1,\dots,K\}}$ be a sequence of i.i.d. Gumbel random variables. Then, we can sample from the discrete variable X by sampling from the Gumbel random variables

$$X = \underset{k\in\{1,\dots,K\}}{\arg\max}(\log \alpha_k + G_k) \tag{5.6}$$

A drawback of this approach is that the argmax operation is not continuous. To address this, a continuous relaxation of the Gumbel-Max Trick has been proposed [43, 60], replacing the argmax operation with a softmax. Note that a discrete random variable can be expressed as a one-hot vector, where the realization of the variable is the index of the non-zero entry. With this notation, a sample from the Gumbel-Softmax relaxation can be expressed by the vector $\hat{X}$ as follows:

$$\hat{X}_k = \text{softmax}\left(\left(\log \alpha_k + G_k\right)/\tau\right) \tag{5.7}$$

where $\hat{X}_k$ is the k^{th} element in $\hat{X}$ and τ is the temperature of the softmax operation. With $\tau \to 0$, the softmax function approaches the argmax function and Equation 5.7 becomes equivalent to the discrete sampler. For $\tau \to \infty$ it becomes a uniform distribution. Since the softmax function is differentiable and G_k is an independent noise term, we can propagate gradients to the probabilities α_k. This enables the gating function to receive learning signals for when to open and when to close. To generate samples, we set the log probabilities to be the output of the gate's first component $\log \alpha = \beta$.

One option to employ the Gumbel-softmax estimator is to use the continuous version from Equation 5.7 during training and obtain discrete samples with Equation 5.6 during testing. An alternative is the *straight-through* version [43] of the Gumbel-softmax estimator. There, during training, for the forward pass we get discrete samples from Equation 5.6, but during the backwards pass we compute the gradient of the softmax relaxation in Equation 5.7. Note that the estimator is biased due to the mismatch between forward and backward pass. However, we observe that empirically the straight-through estimator performs

better and leads to computational graphs that are more category-specific. We illustrate the two different paths during the forward and backward pass in Figure 5.2.

5.2.3 Training Adanets

While training Adanets, we aim to achieve two objectives. First, the main goal is to classify images correctly. As such, the main training loss is a basic multi-class logistic loss that we denote as $\mathcal{L}_{MC}$. Second, we would like the gates to assemble effective network topologies. To this end, the network might need to learn both generalist as well as specialist layers and needs to decide when to use which layer. We achieve this by constraining how often each layer is allowed to be used. This guides the optimization to solutions in which parameters that are relevant to subsets of related categories are grouped together in layers, which minimizes the amount of unnecessary features to be computed. Specifically, we use soft constraints by introducing an additional loss term that encourages each layer to be executed at a certain target rate t. We approximate the execution rates over a mini-batch and penalize deviations from the target rate. Let $\bar{z}_l$ denote the fraction of images within a mini-batch that layer l is executed. Then, the target rate loss is defined as

$$\mathcal{L}_{target} = \sum_{l=1}^{N} (\bar{z}_l - t)^2 \tag{5.8}$$

The target rate provides an easy instrument to adjust the desired computation time. Adanets are robust to a wide range of target rates. We study the effect of the target rate on classification accuracy and inference time in the experimental section. With the multi-class and target-rate losses, the overall training loss is

$$\mathcal{L}_{Adanet} = \mathcal{L}_{MC} + \mathcal{L}_{target} \tag{5.9}$$

We optimize this joint loss with mini-batch stochastic gradient descent.

5.3 Experiments

We perform a series experiments to evaluate the performance of Adanets and whether they learn specialized layers and category-specific computational graphs. Lastly, we look beyond performance and study the robustness of Adanets by analyzing their effect on the susceptibility towards adversarial attacks.

5.3.1 Results on CIFAR

We first perform a set of initial experiments on CIFAR-10 [50] as a proof of concept to validate the gating mechanism and how effectively Adanets distribute computation among layers.

Model configurations and training details

We build an Adanet based on the original Resnet 110 [32]. Besides the additional gating units, the Adanet follows the same architecture as the original Resnet 110. For the gating units, we choose a hidden state of size $d = 16$. The additional gating unit per residual block, adds a fixed overhead of 0.01% more floating point operations and 4.8% more parameters compared to the standard Resnet-110.

We follow a similar training scheme as [32], use momentum with weight

Table 5.1: **Test error on CIFAR 10** in % for Resnet 110 and its adaptive Adanet counterpart. Adanet 110 clearly outperforms Resnet 110 while only using a subset of 85% of the layers. When executing all layers (Adanet 110*), Adanet also outperforms stochastic depth.

Model	**Error**	**#Params** (10^6)	**FLOPs** (10^9)
Resnet 110 [32]	6.61	1.7	0.5
Pre-Resnet 110 [33]	6.37	1.7	0.5
Stochastic Depth Resnet 110 [39]	5.25	1.7	0.5
Adanet 110	5.76	1.78	0.41
Adanet 110*	**5.14**	1.78	0.5

of 0.9 and adopt a weight decay of 5×10^{-4}. All models are trained for 350 epochs with a mini-batch size of 256. We use a step-wise learning rate starting at 0.1 and decaying by 10^{-1} after 150 and 250 epochs. We adopt a standard data-augmentation scheme, where images are padded with 4 pixels on each side, randomly cropped to 32×32 and with probability 0.5 horizontally flipped.

Results

Table 5.1 shows classification results on CIFAR 10 for Resnets [32], pre-activation Resnets [33], stochastic depth [39] and their Adanet counterpart. The table also shows the number of model parameters and floating point operations (multiply-adds). We compare two Adanet variants: For standard Adanets, we only execute those layers with open gates. As a second variant, which we indicate by " * ", we execute all layers and analogous to Dropout [76] and stochastic depth [39] the output of each layer is scaled by its expected execution rate.

From the results, we observe that Adanets clearly outperform their Resnet counterparts, even when using only a subset of the layers. In particular, Adanet 110 with a target-rate of 0.7 uses only 82% of the layers in expectation.

Since Resnet 110 might be over-parameterized for CIFAR-10, the regularization induced by dropping layers could be a key factor to performance. We observe that Adanet 110* outperforms stochastic depth, implying benefits of Adanets beyond regularization. In fact, Adanets learn to identify layers of key importance. In particular, they identify the central role of downsampling layers and learn to always execute them, although they incur computation cost. We do not observe any downward outliers, i.e. layers that are dropped every time.

5.3.2 Results on ImageNet

In a second set of experiments, we evaluate Adanets on ImageNet [17] to study whether they learn to group parameters such that for each input image only relevant features are computed. ImageNet is well suited for this study, as it contains a large set of categories that span a wide variety of concepts including man-made objects, food, as well as many different groups of animals.

Model configurations and training details

We build Adanets based on Resnet 50 and Resnet 101 [32]. The Adanets again follow the same architectures as the original Resnets, with the sole exception of the additional gating units. The size of the hidden state in the gates is again $d = 16$, adding a fixed overhead of 3.9% more parameters and 0.04% more floating point operations. For Adanet 50, all 16 residual layers have gates. For Adanet 101, we fix the early layers up to the second downsampling operation to be always executed. The main reason is that early layers to not yet distinguish between object categories.

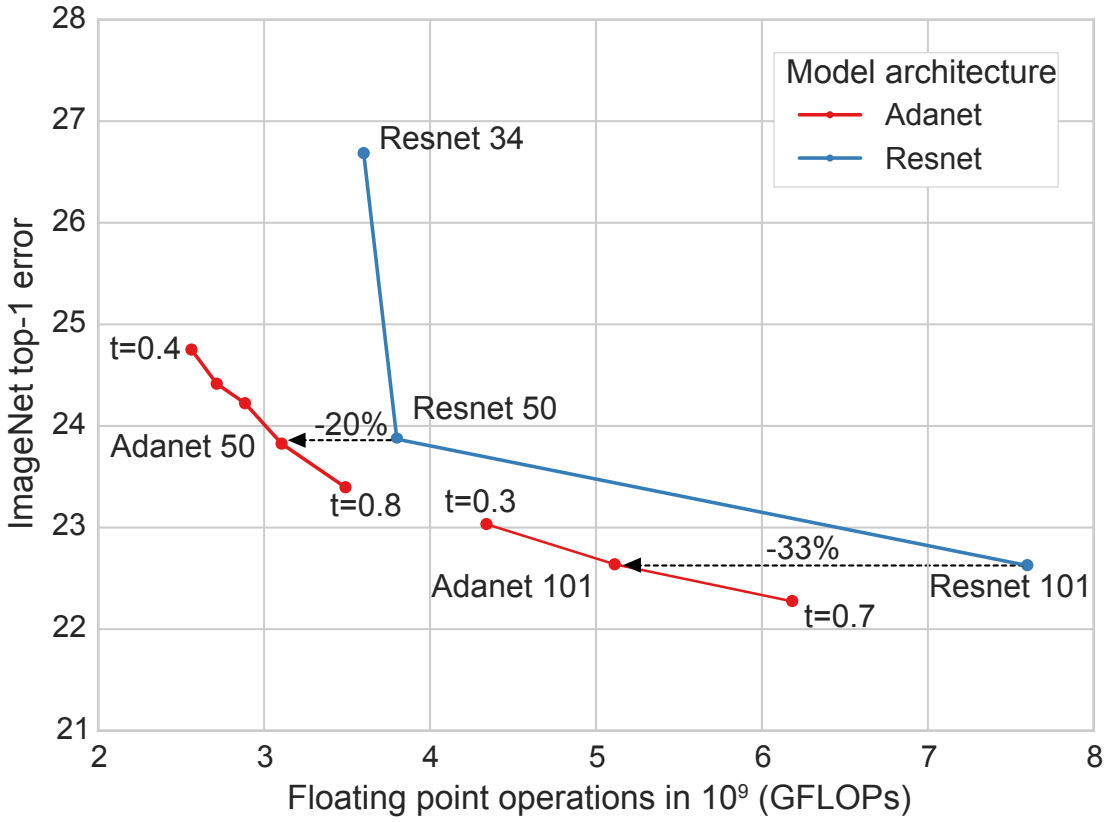


Figure 5.3: **Top-1 accuracy vs. computational cost on ImageNet.** Adanet 50 outperforms Resnet 50, while skipping about 20% of its layers in expectation. Similarly, Adanet 101 outperforms Resnet 101 while requiring 33% less computations. Further, it often appears more effective to decrease the target rate compared to reducing layers in a standard Resnet.

We follow the standard Resnet training procedure, with mini-batch size of 256, momentum of 0.9 and weight decay of 10^{-4}. All models are trained for 100 epochs with step-wise learning rate starting at 0.1 and decaying by 10^{-1} every 30 epochs. We use the data-augmentation procedure as in [32] and at test time first rescale images to 256×256 followed by a 224×224 center crop. The gates are initialized to open at a rate of 85% at the beginning of training.

Quantitative comparison

Figure 5.3 shows top-1 error on ImageNet and computational cost in terms of GFLOPs for Adanet 50, Adanet 101 and the respective Resnets of varying depth. We further show the impact of different target rates on performance and efficiency. In particular, we use target rates from 0.4 to 0.7 for Adanet 50 and 0.3 to 0.5 for Adanet 101. Detailed results including the models' complexities as well as further baselines are presented in Table 5.2.

From the results we make the following key observations. First, both Adanet 50 and Adanet 101 outperform their Resnet counterpart, while at the same time using only a subset of the layers. In particular, Adanet 50 with a target rate of 0.7 uses only about 80% of its layers in expectation. Similarly, Adanet 101 outperforms its respective Resnet while using 33% less computations.

Figure 5.3 also visualizes the effect of the target-rate. As expected, decreasing the target rate reduces computation time. Interestingly, introducing a constraint on the number of executed layers first improves accuracy, before lowering the target rate further decreases accuracy. This demonstrates that Adanets not only improve efficiency, but by grouping and separating parameters, they also improve overall classification quality. Further, it appears often more effective to decrease the target rate compared to reducing layers in a standard Resnet.

Due to surface resemblance, we also compare Adanets to stochastic depth [39]. We observe that, for smaller Resnet models stochastic depth does not provide competitive results on ImageNet. One needs very large models on ImageNet to see benefits from stochastic depth regularization. The paper on

Table 5.2: **Classification error on ImageNet** in % for Adanet 50, Adanet 101 and the respective Resnets of varying depth. Both Adanets outperform their Resnet counterpart, while at the same time using only a subset of the layers. This demonstrates that Adanets are more efficient and also improve overall classification quality.

Model	**Top 1**	**Top 5**	**#Params** (10^6)	**FLOPs** (10^9)
Resnet 34 [32]	26.69	8.58	21.80	3.6
Resnet 50 [32]	24.7	7.8	25.56	3.8
Resnet 50 (our)	23.87	7.12	25.56	3.8
Resnet 101 [32]	23.6	7.1	44.54	7.6
Resnet 101 (our)	**22.63**	6.45	44.54	7.6
Stochastic Depth Resnet 50	27.75	9.14	25.56	3.8
Stochastic Depth Resnet 101	22.80	6.44	44.54	7.6
Adanet 50 [t=0.4]	24.75	7.61	26.56	2.56
Adanet 50 [t=0.5]	24.42	7.42	26.56	2.71
Adanet 50 [t=0.6]	24.22	7.21	26.56	2.88
Adanet 50 [t=0.7]	23.82	7.08	26.56	3.06
Adanet 50 [t=0.8]	**23.40**	**6.79**	26.56	3.49
Adanet 101 [t=0.3]	23.02	6.58	46.23	4.33
Adanet 101 [t=0.5]	22.63	6.26	46.23	5.11
Adanet 101 [t=0.7]	**22.28**	**6.17**	46.23	6.18

stochastic depth [39] reports that even for the very large Resnet 152 performance remains below a basic Resnet. This highlights the opposite goals of Adanets and stochastic depth. Stochastic depth aims to create redundant features by enforcing each subset of layers to model the whole dataset [98]. Adanets aim to separate parameters that are relevant to different subsets of the dataset into different layers.

These results indicates that convolutional networks do not need a fixed feed-forward structure and that Adanets are an effective means to enable adaptive computation graphs that are conditioned on the input image.

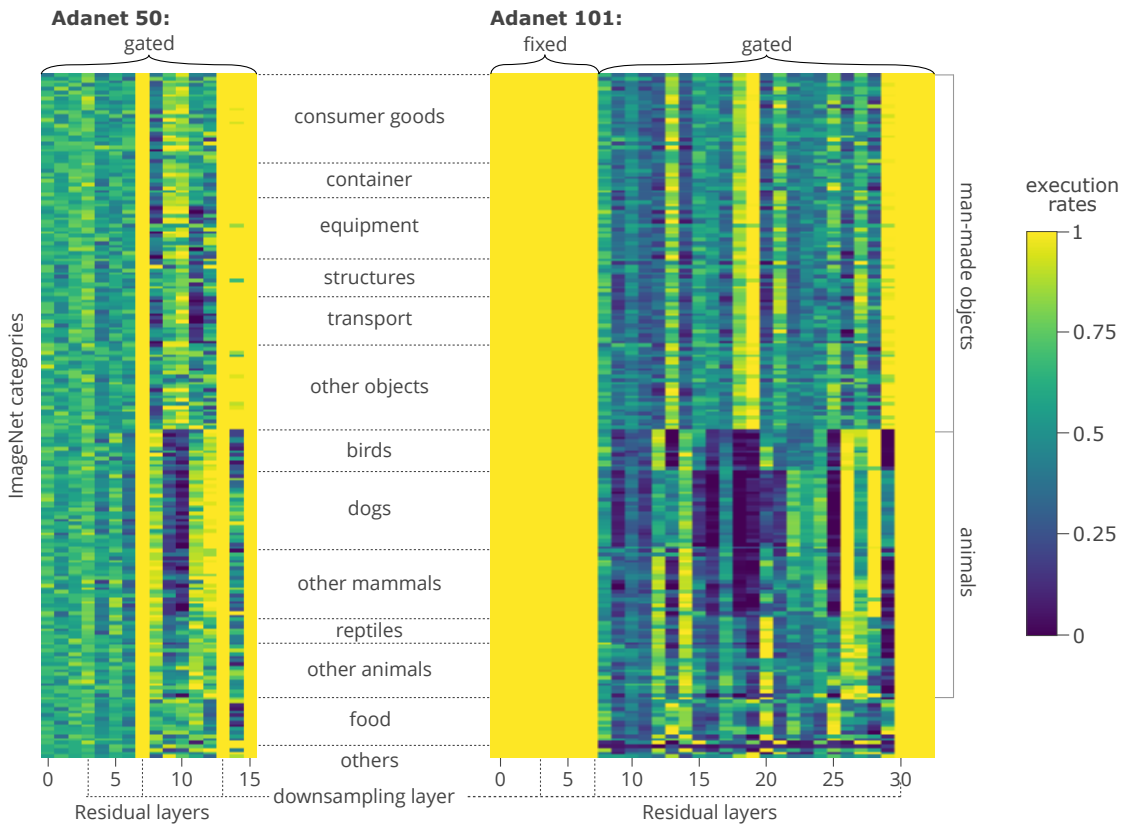


Figure 5.4: **Learned inference paths on ImageNet.**The histograms show for Adanet 50 (left) and Adanet 101 (right) how often each residual layer (x-axis) is executed for each of the 1000 classes in ImageNet (y-axis). We observe a clear difference between the layers used for man-made objects and for animals and even for some mid-level categories such as birds, mammals and reptiles. Without specific supervision, the network discovers parts of the class hierarchy. Further, downsampling layers and the last layers appear of key importance and are executed for all images. Lastly, the left histogram shows that early layers are mostly agnostic to the different classes. Thus, early layers in Adanet 101 are set to be always executed. The remaining layers are sufficient to provide different inference paths for the various categories.

Analysis of learned computational graphs

To analyze the learned inference paths, we study the rates at which different layers are executed for images of different categories. Figure 5.4 shows the execution rates of each layer for Adanet 50 on the left and for Adanet 101 on the

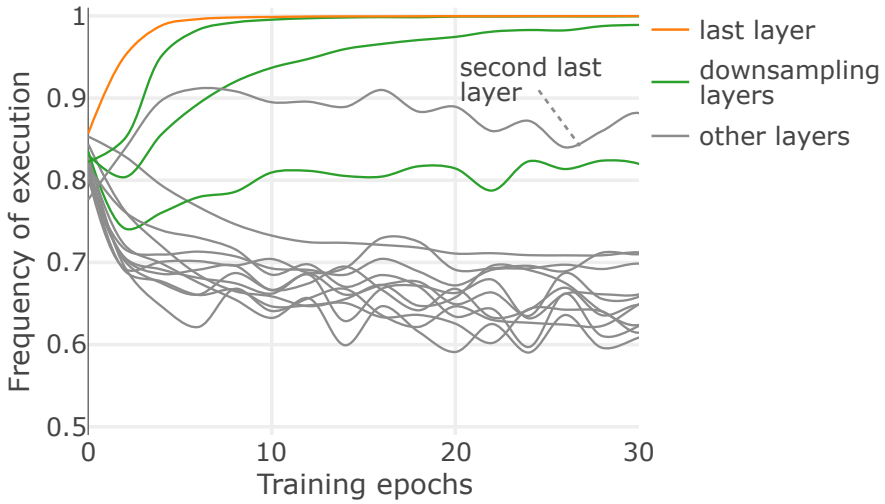


Figure 5.5: **Execution rates per layer over first 30 epochs** of training. The layers are quickly separated into key layers and less critical layers. Downsampling layers and the last layer increase their execution rate, while the remaining layers slowly approach the target rate.

right. The x-axis indicates the residual layers and the y-axis breaks down the execution rates by the 1000 classes in ImageNet. Further, the figure shows high-level and mid-level categories that contain large numbers of classes. The color in each cell indicates the percentage of validation images from a given category that the respective layer is executed.

From the figure, we see a clear difference between man-made objects and animals. Moreover, we even observe distinctions between mid-level animal categories such as birds, mammals, reptiles and other animals. This reveals that the network discovers part of the label hierarchy and groups parameters accordingly. Generally, we observe similar structures in Adanet 50 and Adanet 101. However, the grouping of the mid-level categories is more distinct in Adanet 101 due to the larger number of layers that can capture high-level features. This result demonstrates that Adanets successfully learn layers that focus on specific subsets of categories. It is worthy to note that the training

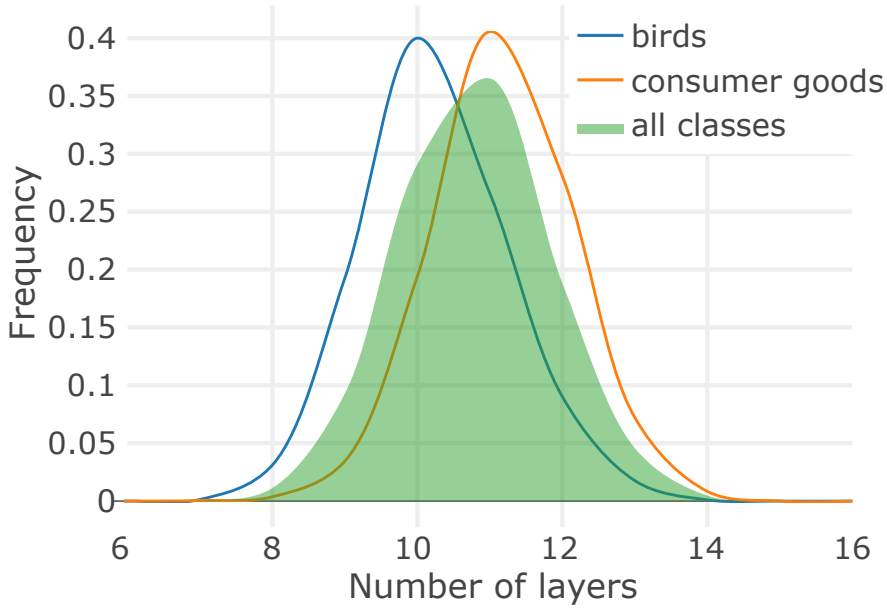


Figure 5.6: **Distribution over the number of executed layers.** For Adanet 50 on ImageNet with target rate 0.4, in average 10.8 out of its 16 residual layers are executed. Images of animals tend to use fewer layers than man-made objects.

objective does not include an incentive to learn category specific layers. The specialization appears to emerge naturally when the computational budget gets constrained.

Further, we observe that downsampling layers and the last layers deviate significantly from the target rate and are executed for all images. This demonstrates their key role in the network (as similarly observed in [98]) and shows how Adanets learn to effectively trade-off computational cost for accuracy.

Lastly, the figure shows that for Adanet 50, inter-class variation is mostly present in the later layers of the network after the second downsampling layer. One reason for this could be that features from early layers are useful for all categories. Further, early layers might not yet capture sufficient semantic information to discriminate between categories. Thus, we keep the early layers

Figure 5.7: **Sample validation images from ImageNet** that use the fewest layers (top) and the most layers (bottom) within the categories of birds, dogs and musical instruments. The examples illustrate how instance difficulty translates into layer usage.

of Adanet 101 fixed to be always executed. The remaining layers still provide sufficient flexibility for different inference paths for the various categories.

Figure 5.5 shows a typical trajectory of the execution rates during training for Adanet 50. The layers are initialized to execute a rate of 85% at the start of training. The figure shows the first 30 training epochs and highlights how the layers are quickly separated into key layers and less critical layers. Important layers such as downsampling and the last layers increase their execution rate, while the remaining layers slowly approach the target rate.

Variable inference time

Due to the adaptive inference paths, computation time varies across images. Figure 5.6 shows on the left the distribution over how many of the 16 residual layers in Adanet 50 are executed over all ImageNet validation images. On average 10.81 layers are executed with a standard deviation of 1.11. The figure also shows the distributions for the mid-level categories of birds and consumer goods. In expectation, images of birds use one layer less than images of con-

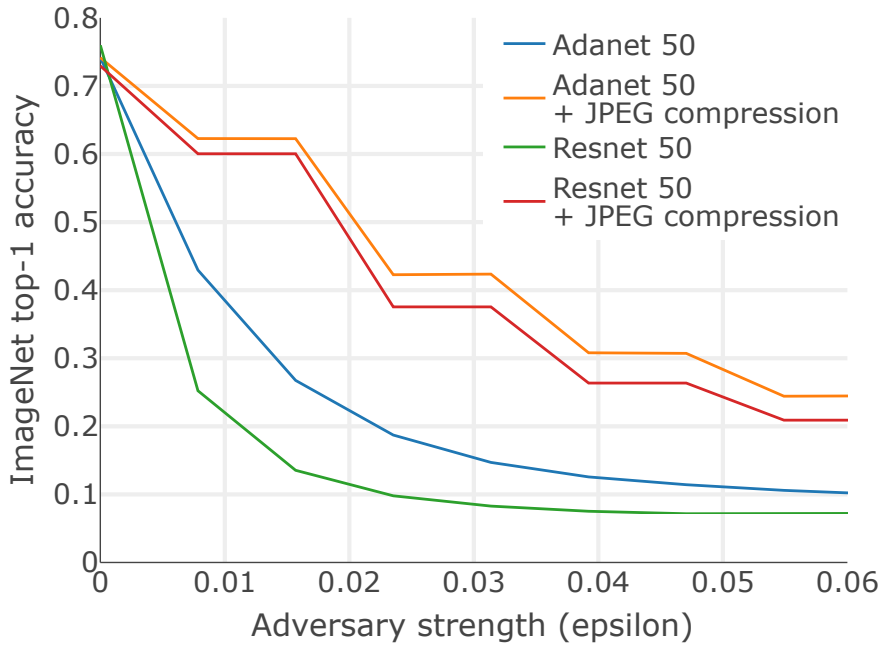


Figure 5.8: **Adversarial attack using Fast Gradient Sign Method.** Adanets are consistently more robust than plain Resnets, independent of adversary strength. The additional robustness persists even when applying additional defense mechanisms.

sumer goods. Further, from Figure 5.4 we also know that the two groups not only use different numbers of layers, but also different sets of layers. Figure 5.7 shows samples from the validation images that use the fewest and the most layers within the categories of birds, dogs and musical instruments. The examples highlight that easy instances with iconic views require only a few layers. Difficult instances that are small or occluded need more computation.

5.3.3 Robustness to adversarial attacks

In a third set of experiments we aim to understand the effect of adaptive computation graphs on the susceptibility towards adversarial attacks. On one hand, if adversarial perturbations change the computational graph such that key lay-

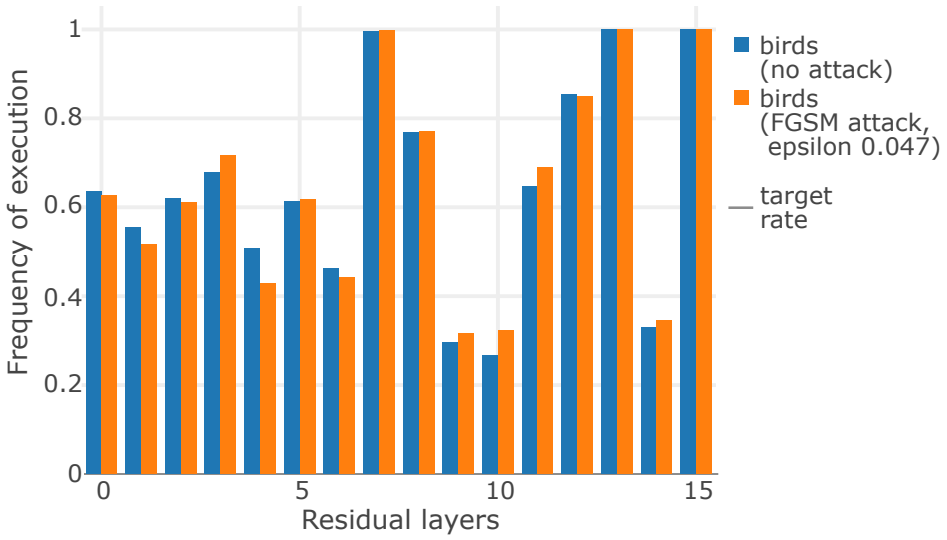


Figure 5.9: **Effect of adversarial attack on gating functions.** Average execution rates per layer for images of birds before and after the attack. The execution rates remain mostly unaffected by the attack.

ers of the network are skipped, performance might degrade. On the other hand, the stochasticity of the graph might improve robustness.

To shed light on this question, we perform a Fast Gradient Sign Attack [26] on Resnet 50 and Adanet 50, both trained on ImageNet. The results are presented in Figure 5.8. In the graph on the left side, on the x-axis we show the strength of the adversary as measured in the amount each pixel is allowed to be changed (epsilon). On the y-axis we report the top-1 classification accuracy on ImageNet. We observe the Adanet 50 is consistently more robust, independent of adversary strength. To investigate whether this additional robustness complements other known defenses [29], we perform a follow-up experiment in which we perform JPEG compression on the adversarial examples. We follow [29] and use a JPEG quality setting of 75%. While both networks greatly benefit from the JPEG compression defense, Adanet 50 remains more robust, indicating that the additional robustness can complement other defenses.

To understand the effect of the attack on the gating mechanism, we look at the execution rates before and after the attack. On the right side, Figure 5.9 shows the average execution rates per layer over all bird categories for Adanet 50 before and after performing an FGSM attack with epsilon 0.047. The results show that, although the accuracy of the network drops from 74.62% to 11%, execution rates remain similar. The resilience of the gating towards the attack might arise for multiple reasons. First, the stochasticity induced by the Gumbel noise might outweigh the noise introduced by the attack. Further, the global average pooling operation might cancel out some of the adversarial perturbations.

5.4 Conclusion

In this work, we have shown that convolutional networks do not need fixed feed-forward structures. With Adanets, we have introduced Convnets that adaptively assemble their network graph on the fly based on the input image. We have shown in experiments on ImageNet that Adanets group parameters for related classes into specialized layers and learn to only execute those layers relevant to the input. This allows decoupling inference time from the number of learned concepts and improves both efficiency as well as overall classification quality.

This work opens up numerous paths for future work. For example, with respect to network architecture, it would be intriguing to extend this work beyond Resnets to other high-level architectures such as densely-connected [38] or inception-based [81] networks. Further, it would be interesting to investigate

Adanets for other tasks such as image segmentation or object detection. From an optimization perspective, we have seen that layers take on different roles in Adanets. Future work could study whether one should treat individual layers differently during training to further improve their effectiveness. From a practitioner's point of view, it might be exciting to extend this work into a framework where the set of executed layers is adaptive, but their number is fixed so as to achieve constant inference times. Lastly, we have seen that the gates are largely unaffected by basic adversarial attacks. For an adversary, it could be interesting to investigate attacks that specifically target the gating functions.

CHAPTER 6

CONCLUSIONS

In this dissertation, we studied conditional models for computer vision. A key motivation for this line of work is that previous vision models are largely agnostic to the context in which they are used, limiting the usefulness of their results. To address this shortcoming, we presented multiple models that can incorporate contextual information to provide results tailored to a given context. First, we presented an image tagging and retrieval system that can personalize predictions to individual users, by learning from their prior usage of hashtags. We then focused on the task of visual similarity search and introduced Conditional Similarity Networks. When searching for images similar to an input image, the proposed model allows users to specify according to what aspect images should be similar. Lastly, we focused on the underlying convolutional networks themselves and showed how they can benefit from taking the context into account, in which they are used. In particular, we presented Adanets, convolutional networks that can define their network topology conditioned on the input image. By learning both general layers useful to all images and expert layers specializing on subsets of categories, Adanets improve both efficiency and also overall classification quality.

This work opens up numerous paths for future work. Collecting labeled training data is often one of the main bottlenecks for building vision models. Weakly annotated data obtained from the web provides an interesting alternative. However, using web data as supervision comes with the challenge that it is often only informative in the context it was created. By allowing labels to carry multiple different meanings, conditional vision models could provide an im-

portant means towards training from noisy data in the wild and thus alleviate the need for expensive annotations.

Further, it would be interesting to study how to extend the presented models to incorporate prior knowledge more generally. For the task of image tagging, we showed how to include user profiles to personalize results. While this can capture basic preferences of a user, it would be interesting to use further prior knowledge, such as the current location of the user or what the user is currently doing. Similarity, for the proposed Adanets we have shown how to design the network topology based on the current input image. Considering the scenario of analyzing a video, after processing a set of frames one might already have a good understanding of what is happening in the current scene. It would be interesting to study how to incorporate this information when designing the network for later frames.

Another intriguing direction would be to discover different notions of similarity without specific supervision. For conditional similarity networks, each training example needs to be associated with one of the pre-defined notions of similarity. The user-conditional hashtag model, removes the need for a list of pre-defined aspects, but the image-label pairs still need to be related to specific users. The proposed Adanets provide a first glimpse into how an unsupervised approach might work. By learning which subset of layers to choose for each image, the network discovers part of the label hierarchy without any special supervision. A similar approach might also be able to discover and separate different notions of similarity.

BIBLIOGRAPHY

[1] Ehsan Amid and Antti Ukkonen. Multiview triplet embedding: Learning attributes in multiple maps. In *International Conference on Machine Learning (ICML)*, 2015.

[2] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *Journal of Machine Learning Research (JMLR)*, 15:2773–2832, 2014.

[3] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. *Proceedings of NAACL-HLT*, 2016.

[4] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] Boris Babenko, Steve Branson, and Serge Belongie. Similarity metrics for categorization: from monolithic to category specific. In *International Conference on Computer Vision (ICCV)*, 2009.

[6] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *International Conference on Machine Learning (ICML)*, 2004.

[7] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

[8] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[9] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[10] Zoya Bylinskii, Sami Alsheikh, Spandan Madan, Adria Recasens, Kimberli Zhong, Hanspeter Pfister, Fredo Durand, and Aude Oliva. Understanding infographics through textual and visual tag prediction. *arXiv preprint arXiv:1709.09215*, 2017.

[11] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale online learning of image similarity through ranking. *The Journal of Machine Learning Research (JMLR)*, 11:1109–1135, 2010.

[12] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[13] François Chollet. Information-theoretical label embeddings for large-scale image classification. *arXiv preprint arXiv:1607.05691*, 2016.

[14] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[15] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *International Conference on Image and Video Retrieval (CIVR)*, 2009.

[16] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning (ICML)*, 2008.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[18] Emily Denton, Jason Weston, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. User conditional hashtag prediction for images. In *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015.

[19] Harris Drucker, Corinna Cortes, Lawrence D. Jackel, Yann LeCun, and Vladimir Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994.

[20] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):10, 2011.

[21] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan

Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[22] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[23] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[24] Fréderic Godin, Viktor Slavkovikj, Wesley De Neve, Benjamin Schrauwen, and Rik Van de Walle. Using topic models for twitter hashtag recommendation. In *International Conference on World Wide Web (WWW)*, 2013.

[25] Wenjuan Gong, Michael Sapienza, and Fabio Cuzzolin. Fisher tensor decomposition for unconstrained gait recognition. In *ECML/PKDD Workshops*, 2013.

[26] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[27] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[28] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*. Number 33. US Govt. Print. Office, 1954.

[29] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[30] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[31] Richard A Harshman and Margaret E Lundy. Parafac: Parallel factor analysis. *Computational Statistics & Data Analysis*, 18(1):39–72, 1994.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*, 2016.

[34] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*, 1991.

[35] G. Van Horn and P. Perona. The devil is in the tails: Fine-grained classification in the wild. 2017.

[36] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[37] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense convolutional networks for efficient prediction. *arXiv preprint arXiv:1703.09844*, 2017.

[38] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[39] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision (ECCV)*, 2016.

[40] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *International Conference on Computer Vision (ICCV)*, 2017.

[41] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[42] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[43] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization

with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

[44] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. *International Conference on Computer Vision (ICCV)*, 2017.

[45] Armand Joulin, Laurens van der Maaten, Allan Jabri, and Nicolas Vasilache. Learning visual features from large weakly supervised data. In *European Conference on Computer Vision (ECCV)*. Springer, 2016.

[46] Theofanis Karaletsos, Serge Belongie, and Gunnar Rätsch. Bayesian representation learning with oracle constraints. In *International Conference on Learning Representations (ICLR)*, 2016.

[47] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[49] I Krasin, T Duerig, N Alldrin, A Veit, S Abu-El-Haija, S Belongie, D Cai, Z Feng, V Ferrari, V Gomes, et al. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github. com/openimages*, 3, 2016.

[50] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[52] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[53] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[54] A. Li, A. Jabri, A. Joulin, and L.J.P. van der Maaten. Learning visual n-grams from web data. In *International Conference on Computer Vision (ICCV)*, 2017.

[55] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[56] Jiwei Li and Dan Jurafsky. Do multi-sense embeddings improve natural language understanding? *Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2015.

[57] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying neural style transfer. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[58] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.

[59] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *International Conference on Computer Vision (ICCV)*, 2015.

[60] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

[61] Jitendra Malik and Pietro Perona. Preattentive texture discrimination with early vision mechanisms. *Journal of the Optical Society of America*, 1990.

[62] Brian McFee and Gert Lanckriet. Learning multi-modal similarity. *The Journal of Machine Learning Research (JMLR)*, 12:491–523, 2011.

[63] Ishan Misra, Abhinav Gupta, and Martial Hebert. From red wine to red tomato: Composition with context. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[64] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *International Conference on Machine Learning (ICML)*.

[65] Tye Rattenbury, Nathaniel Good, and Mor Naaman. Towards automatic extraction of event and place semantics from flickr tags. In *International Conference on Research and Development in Information Retrieval (SIGIR)*, 2007.

[66] Scott Reed, Kihyuk Sohn, Yuting Zhang, and Honglak Lee. Learning to disentangle factors of variation with manifold interaction. In *International Conference on Machine Learning (ICML)*, 2014.

[67] Steffen Rendle. Factorization machines. In *International Conference on Data Mining (ICDM)*, 2010.

[68] Robert E Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[69] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[70] Thomas Serre, Aude Oliva, and Tomaso Poggio. A feedforward architecture accounts for rapid categorization. *Proceedings of the National Academy of Sciences*, 104(15):6424–6429, 2007.

[71] Amnon Shashua and Tamir Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *International Conference on Machine Learning (ICML)*, 2005.

[72] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representation (ICLR)*, 2017.

[73] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[74] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[75] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard

Schölkopf. Large scale multiple kernel learning. *The Journal of Machine Learning Research (JMLR)*, 7:1531–1565, 2006.

[76] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research (JMLR)*, 15(1):1929–1958, 2014.

[77] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[78] C. Stanley and M.D. Byrne. Comparing vector-based and bayesian memory models using large-scale datasets: User-generated hashtag and tag prediction on twitter and stack overflow. *Psychological Methods*, 21(4):542–565, 2016.

[79] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *International Conference on Computer Vision (ICCV)*, 2015.

[80] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *International Conference on Computer Vision (ICCV)*, 2017.

[81] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[82] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[83] Omer Tamuz, Ce Liu, Ohad Shamir, Adam Kalai, and Serge J Belongie. Adaptively learning the crowd kernel. In *International Conference on Machine Learning (ICML)*, 2011.

[84] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *Conference on-Pattern Recognition (ICPR)*, 2016.

[85] Joshua B Tenenbaum and William T Freeman. Separating style and content. In *Advances in Neural Information Processing Systems (NIPS)*, 1997.

[86] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[87] L.R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[88] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research (JMLR)*, 9(2579-2605):85, 2008.

[89] Laurens Van der Maaten and Geoffrey Hinton. Visualizing non-metric similarities in multiple maps. *Machine learning*, 87(1):33–55, 2012.

[90] Laurens Van Der Maaten and Kilian Weinberger. Stochastic triplet embedding. In *Workshop on Machine Learning for Signal Processing (MLSP)*, 2012.

[91] M Alex O Vasilescu. Human motion signatures: Analysis, synthesis, recognition. In *International Conference on Pattern Recognition (ICPR)*, 2002.

[92] M Alex O Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision (ECCV)*, 2002.

[93] Andreas Veit, Neil Alldrin, Gal Chechik, Ivan Krasin, Abhinav Gupta, and Serge Belongie. Learning from noisy large-scale datasets with minimal supervision. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[94] Andreas Veit and Serge Belongie. Convolutional networks with adaptive computation graphs. *arXiv preprint arXiv:1711.11503*, 2017.

[95] Andreas Veit, Serge Belongie, and Theofanis Karaletsos. Conditional similarity networks. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[96] Andreas Veit, Balazs Kovacs, Sean Bell, Julian McAuley, Kavita Bala, and

Serge Belongie. Learning visual clothing style with heterogeneous dyadic co-occurrences. In *International Conference on Computer Vision (ICCV)*, 2015.

[97] Andreas Veit, Maximilian Nickel, Serge Belongie, and Laurens van der Maaten. Separating self-expression and visual content in hashtag supervision. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[98] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[99] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision (IJCV)*, 57(2):137–154, 2004.

[100] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[101] Weiran Wang, Raman Arora, Karen Livescu, and Jeff Bilmes. On deep multi-view representation learning. In *International Conference on Machine Learning (ICML)*, 2015.

[102] Peter Welinder, Steve Branson, Pietro Perona, and Serge J Belongie. The multidimensional wisdom of crowds. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[103] J. Weston, S. Bengio, and N. Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine Learning*, 81(1):21–35, 2011.

[104] Michael Wilber, Iljung S Kwak, David Kriegman, and Serge Belongie. Learning concept embeddings with combined human-machine expertise. In *International Conference on Computer Vision (ICCV)*, 2015.

[105] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[106] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How trans-

ferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[107] Aron Yu and Kristen Grauman. Fine-grained visual comparisons with local learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[108] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2014.